



EMu Documentation

Release Notes: **EMu 4.3**

Document Version 1

EMu Version 4.3

EMu
Museum
Management
System



kesoftware.com
©2014 KE Software
All rights reserved

Contents

Here you will find collected together the Release Notes for EMu 4.3, alongside all documents referenced in the notes. These release notes and documents are also available on the [EMu website](#).

This PDF document brings together a number of individually published documents: please note that page numbering below refers to this combined PDF document and not to the page numbers printed at the bottom of pages, as each individual document has its own internal numbering:

Release Notes: EMu 4.3	5
Scheduled Operations	21
EMu GUID Support	62

Release Notes: EMu 4.3

Release Date: 07 August 2014

Requirements

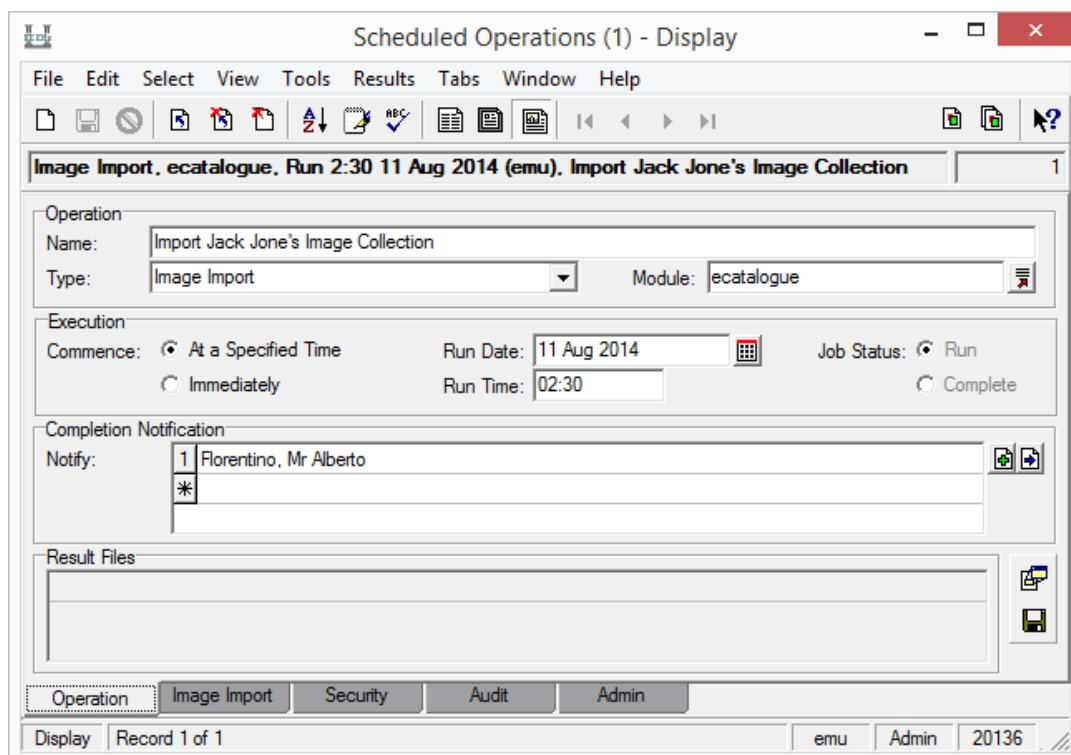
- Windows 2003, Vista, Windows 7, Windows 8, Windows 8.1
- [Texpress 8.3.013](#) or later
- [TexAPI 6.0.011](#) or later
- [Perl 5.8.8](#) or later

New Features

Scheduled Operations

The Scheduled Operations facility enables the scheduling of time and computing intensive operations to be run immediately or at a specified date and time. An operation is defined by:

- The type of operation to run (e.g. delete records)
- The module to which the operation applies
- A time and date to commence the operation
- A list of people to notify when the operation is complete



A scheduled operation is defined and stored as a record in the new Scheduled Operations module (eoperations). Any files created when the operation is executed are listed in the *Result Files* table on the Operation tab. The following three type of the operations are supported:

- Merge records
- Delete records
- Image import

System administrators may define their own operations. A complete description of the support for Scheduled Operations can be found in the Scheduled Operations documentation.

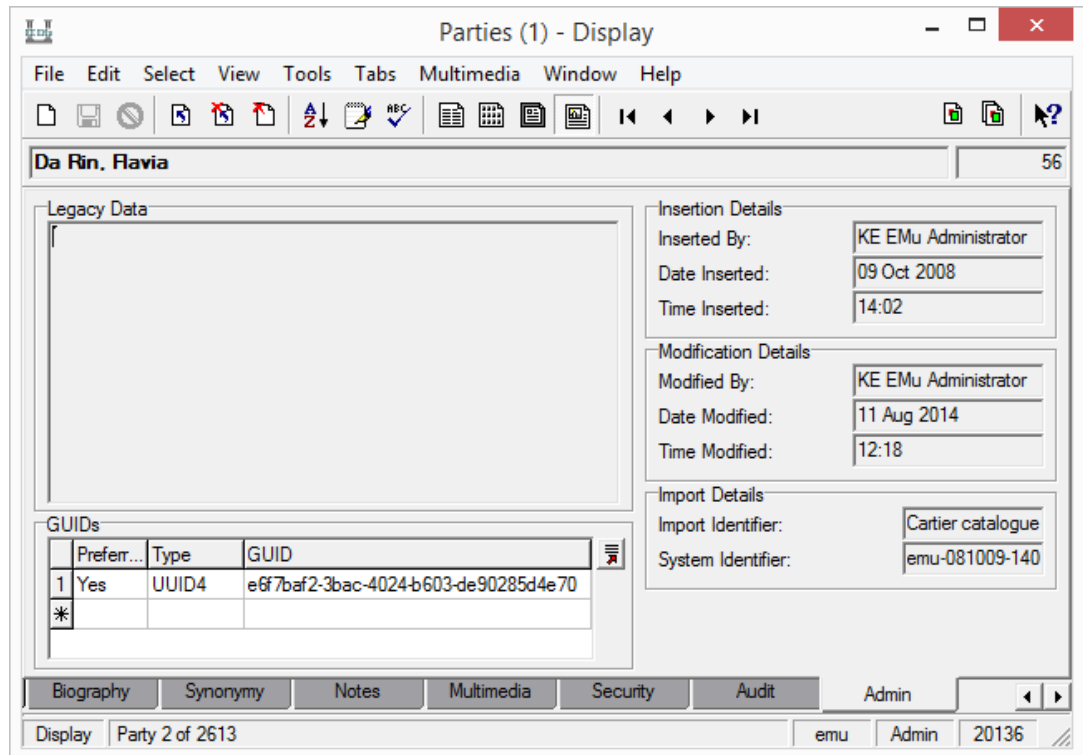
GUID / UUID support

A Globally Unique Identifier (GUID) is a persistent unique reference number used as an identifier in computer software. Increasing global initiatives in data sharing require the use of a unique identifier for each discrete bit of data (record). The implementation of GUIDs provides:

- Storage for a list of GUIDs in modules.
- Automatic GUID generation when a record is saved.

Almost all modules have had support added for GUIDs. The exceptions are modules that store audit based information (e.g. Audit Trails) or system information (e.g. Registry). The GUID table is located on the Admin tab and consists of the following value:

- *Preferred* - which entry in the list of GUIDs is the preferred value.
- *Type* - the type of the GUID value (e.g. UUID4).
- *GUID* - the GUID value itself.



GUID values can be auto-generated when a record is saved. Registry entries are used to determine which tables require auto-generated GUIDs and what type should be used when generating the GUID. Currently, GUIDs are auto-generated in compliance with UUID Version 4. An example of a UUID 4 GUID is 84b567d5-dbbd-468a-be12-770747ebc397.

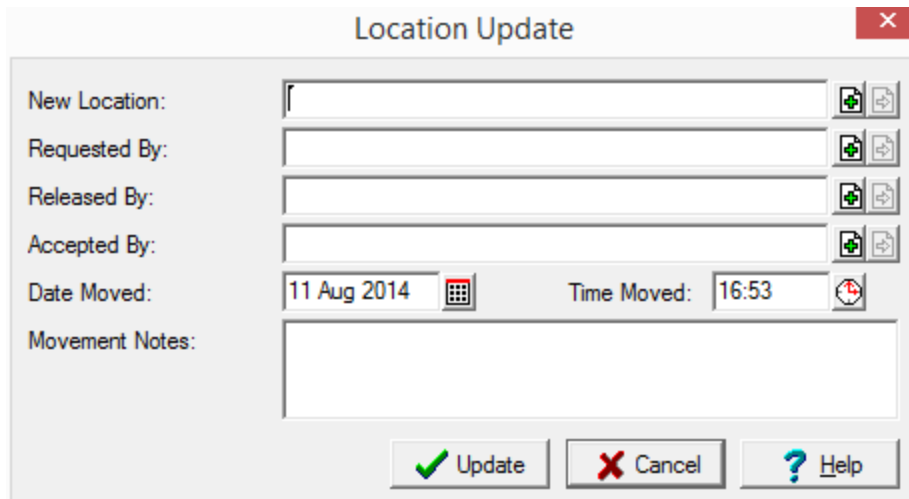
A complete description of the support provided for GUIDs can be found in the [EMu GUID Support](#) documentation.

Improvements

Calendar controls for batch updates

The batch update tools have been updated to include the calendar control introduced in EMu 4.2. The calendar controls can be found on the following update dialogue boxes:

- Condition Update
- Location Update
- Taxon Update
- Valuation Update



The screenshot shows a 'Location Update' dialog box with the following fields and controls:

- New Location:** Text input field with a green plus icon and a right-pointing arrow icon.
- Requested By:** Text input field with a green plus icon and a right-pointing arrow icon.
- Released By:** Text input field with a green plus icon and a right-pointing arrow icon.
- Accepted By:** Text input field with a green plus icon and a right-pointing arrow icon.
- Date Moved:** Text input field containing '11 Aug 2014' and a calendar icon.
- Time Moved:** Text input field containing '16:53' and a clock icon.
- Movement Notes:** A large empty text area for notes.
- Buttons:** 'Update' (green checkmark), 'Cancel' (red X), and 'Help' (blue question mark).

Image Magick upgraded

ImageMagick, which is used by EMu to view and manipulate images, has been upgraded to the latest version (6.8.8). The upgrade provides support for a number of new image formats and fixes a number of issues. In particular, improved support for DNG (digital negatives) and PDF (portable document format) formats is provided.

Issues Resolved

Issue	Resolution
<p>The output file for XSLT based reports is saved in files with a .html extension. If the output type specified in the report via <xsl:output> is not HTML, then the file extension should reflect the output type specified.</p>	<p>The extension of the file created for an XSLT based report is based on the type specified in the <xsl:output> tag.</p>
<p>The label name of the multimedia thumbnail in List View in the Multimedia module is shown as <i>MulDocumentType</i>. The title is displayed regardless of the display language selected. The label should be changed to <i>Multimedia Thumbnail</i> for English and an appropriate translation for other languages.</p>	<p>The label has been changed from <i>MulDocumentType</i> to <i>Multimedia Thumbnail</i>. Translations have been added for other languages.</p>
<p>The Summary Data and Extended Data calculations for the Collections Description module do not handle multiple languages correctly. The data generated is correct where a single language system is used.</p>	<p>The Summary Data and Extended Data calculations for the Collections Description module now handles multiple languages correctly.</p>
<p>The Global Replace facility does not allow a global edit to be performed on a hierarchy where the level being changed is read-only. The limitation ensures that only existing hierarchy combinations may be used in a replacement. The restriction means that global replacements cannot be used to change a read-only hierarchy to a different existing combination. Ideally it should be possible to alter the value in a hierarchy where the level being changed is read-only.</p>	<p>The value in a hierarchy where the level being changed is read-only can now be changed. If the value would form a new hierarchy combination, then the Global Replace is still allowed. The new hierarchy will be added to the Lookup List.</p>
<p>An invalid server-side schema file may be produced if a field's prompt has a single quote character in it. The field prompt is defined in the database rather than the prompt displayed in the Windows Client. A correct schema file should be generated even if a field's prompt contains a single quote.</p>	<p>A valid schema file is now generated even if a field's prompt contains a single quote character.</p>

Issue	Resolution
<p>The Windows client installer program does not include the version information in the <i>Add/Remove Programs</i> section of the Windows Control Panel. Some desktop roll-out software uses the version information to determine which software packages need to be upgraded on a given computer. The version number should be in the same format as that used with EMu releases.</p>	<p>Version information is now added to the EMu entry in the <i>Add/Remove Programs</i> section of the Windows Control Panel.</p>
<p>The Windows client creates missing ODBC Data Sources for each module when it is invoked. If an ODBC Data Source is damaged, that is key entries are missing, the entry is not repaired. A check for damaged entries which can then be repaired should be performed.</p>	<p>Damaged ODBC Data Source entries are now repaired when the Windows client is invoked.</p>
<p>When the selected row in a LinkGrid control with RichEdit controls associated with it changes, the cursor position in the associated RichEdit control is not reset to the beginning of the field. The cursor should be placed at the start of the field when the row in the grid changes.</p>	<p>The cursor in a RichEdit control associated with a LinkGrid is now placed at the beginning of the field when the selected row in the grid changes.</p>
<p>The Lookup List server (lutserver) is used to maintain the values in Lookup Lists. When a hierarchy that contains multiple values at each level is checked, the server may be slow to check that each combination exists as a Lookup combination.</p>	<p>The time taken to check hierarchies containing multiple values at each level has been improved significantly.</p>
<p>The server-side Registry manipulation programs (emuregload, emuregdelete and emuregupdate) do not correctly handle Registry entries over 8192 characters long. The error messages emitted may be somewhat cryptic.</p>	<p>The Registry manipulation programs have been enhanced to handle infinite length Registry entries. Better error messages are displayed and a new program emuregcheck has been added to allow the format of existing Registry entries to be checked.</p>
<p>Server-side scripts that are executed on behalf of a user should produce output in the language the user last used in the Windows client. A new Registry entry specifying a user's prompt language should be added. The entry can then be checked by scripts and used to produce output in the correct language.</p>	<p>A new Registry entry containing the language last used by a user has been added. The entry is updated automatically by the Windows client and is used by server-side scripts to produce output in the correct language.</p>

Issue	Resolution
<p>The <code>System Client Version Registry</code> entry does not allow the release date to be included as part of the minimum supported version number. For example, an entry of <code>4.2</code> allows all version 4.2 clients to be invoked. In some instances a finer grain of control is required. For example a value of <code>4.2 (1404031)</code> would block all clients before version 4.2 (1404031).</p>	<p>The <code>System Client Version Registry</code> entry has been changed to <code>System Setting Client Version</code> and support for release dates has been added.</p>
<p>If the Calendar pop-up is used to select a date for performing a query, then the value placed in the field is not wrapped in double quotes. If the date format is set to <code>dd MMM yyyy</code>, then the search generated will treat the date as three separate components rather than a single date value.</p>	<p>When a date is selected from a Calendar pop-up that is to be used in a search, the date is now enclosed in double quotes, ensuring a single date value is used.</p>
<p>The Narratives module and other modules containing HTML based editing may not change to Edit mode when the HTML version of the data is modified. The issue only arises where Internet Explorer 11 is installed on the user's computer.</p>	<p>When HTML based data is modified, the module is now placed into Edit mode for computers with Internet Explorer 11 installed.</p>
<p>Export records created via the Scheduled Exports facility cannot be deleted. The Export records were meant to provide a complete audit of all data exported and so cannot be deleted. However, due to space considerations, deletion of older Export records would be useful.</p>	<p>Export records may now be deleted.</p>
<p>The <code>fifoserver</code> does not output Unicode (UTF-8) characters correctly when logging calls and outputting results. Non-Unicode based systems (e.g. Latin 1) are not affected.</p>	<p>The correct Unicode characters are now output by the <code>fifoserver</code>.</p>
<p>The title displayed below thumbnails in Contact Sheet mode may be incorrect if the order of the matching records has been modified in List mode. The issue is caused by the Contact Sheet cache not being cleared when records are re-ordered.</p>	<p>The correct title is now displayed below thumbnails in Contact Sheet mode when records have been re-ordered.</p>

Issue	Resolution
<p>The server-side Lookup List rebuild program <code>emulutsrebuild</code> generates an error on Unicode based systems when a Lookup List value is not encoded correctly. Since the data is already in the system, albeit incorrectly, <code>emulutsrebuild</code> should generate a warning and continue processing data.</p>	<p><code>emulutsrebuild</code> now generates a warning and continues processing data if an invalid Unicode sequence is encountered.</p>
<p>The error message Column "irn_1" is read only - primary key. in Column irn may be displayed after a number of modifications have been made to existing records. The message does not appear when creating new records. The appearance of the error message is somewhat random.</p>	<p>The error message is no longer displayed when changing existing records.</p>
<p>The list of values in a given Lookup List may not be correct when the field is part of a hierarchy containing a double nested grid. A double nested grid is a LinkGrid control that is associated with another LinkGrid control. Selecting a row in an associated grid alters the contents of the LinkGrid control. The values displayed are generally a sub-set of the allowable values.</p>	<p>The list of values displayed for a Lookup List in a double nested grid is now the complete set of values.</p>
<p>Audit records created with ISO-8859-1 (Latin 1) characters as part of the data may not display correctly in the Audit Trails module. The Latin 1 characters are converted to Unicode (UTF-8) characters and stored in the Audit table. The conversion should not occur.</p>	<p>Audit records containing Latin 1 characters are now stored and displayed correctly in the Audit Trails module.</p>
<p>The Microsoft Visual Basic error message Run-time error '1004': This operation cannot be done because the data is refreshing in the background. may be displayed when running the Data Dictionary report in the Field Help module. The error may also occur when other Microsoft Excel based reports are invoked.</p>	<p>The error message is no longer displayed when the Data Dictionary report in the Field Help module is run.</p>
<p>The email notifications generated nightly by the server-side <code>emunotify</code> script may not be sent if the data in the email notification contains a closing bracket ')' at the end of a line.</p>	<p>The email notifications are now generated and posted regardless of the data in the notification message.</p>

Issue	Resolution
The error message Invalid selection has been made. Please use the Lookup List may appear when a new insertion is commenced. The error only occurs if the module has a read-only hierarchy of Combo Boxes and one of the Combo Boxes has the AllowEmpty property set to false.	The error message is no longer displayed when a new insertion is commenced.
The <i>Quality</i> field in the <i>Image Attributes</i> group on the Resolutions tab of the Multimedia module is not populated when the Multimedia>Generate Resolution>Selected Records command is selected from the Menu bar.	The <i>Quality</i> field is now populated when the generate resolutions command is invoked.
If the Edit>Ditto>All Fields command is invoked in the <i>Value (Edit)</i> field in the Registry module, then only the <i>Value</i> fields are dittoed. The <i>Value (Edit)</i> field remains empty.	All fields are now dittoed correctly.
If a sort is executed in a LinkGrid control that contains more than two rows, the data in some cells may appear as though it is not sorted. If the screen is repainted (by covering the grid and then uncovering it), the grid is drawn correctly.	The data in the LinkGrid control is now displayed correctly after the data is sorted.
The <i>Audit Record</i> field on the Summary tab in the Audit Trails module does not resize when the form size is increased. The field remains the same size as it was when the module was invoked.	The <i>Audit Record</i> field now resizes correctly when the form size is increased.
If a selection of records is copied from List View and then pasted into Microsoft Outlook, the data does not appear in a formatted table. The data should appear in an HTML based table.	Data copied from List View and pasted into Microsoft Outlook now appears as a formatted table.
If one or more records are copied from List View and the first column of data copied contains empty values, the data in rows where the first column is empty is moved to the left by one column when the records are pasted into Microsoft Outlook or Word.	Rows where the data in the first column is empty are now pasted correctly, that is with an empty first column.
The <i>Quality</i> field in the <i>Image Attributes</i> group on the Resolutions tab of the Multimedia module may be populated with incorrect data when the <code>Update Resources</code> command is used to populate the value.	The correct value is now stored in the <i>Quality</i> field when updated via the <code>Update Resources</code> command.

Issue	Resolution
All database generated error messages contain UTF-8 characters where non-ASCII characters are required. If a client has an ISO-8859-1 based system, the error message will not display correctly in the Windows client. The error message is displayed correctly in Unicode (UTF-8) based systems.	All database error messages are now displayed with the correct encoding, regardless of the system character-set used.
The data stored in the <i>Audit Record</i> field on the Summary tab in the Audit Trails module cannot be copied onto the Windows clipboard. If a user selects the data (via CTRL+A) and then copies it (via CTRL+C), the data cannot be pasted (via CTRL+V) into another application.	The data stored in the Audit Record field can now be copied onto the Windows clipboard.
The error message Cannot locate http://... resource on server may appear when the Multimedia>Launch Viewer command is invoked. The page is opened correctly if the Launch Viewer button on the Multimedia Toolbar is selected or the image in the Image pane is double clicked.	The error message no longer occurs and the correct page is displayed when the Multimedia>Launch Viewer command is invoked.
If data is entered into the second or subsequent rows of the <i>Notes Summary</i> table on the Notes tab in the Conservation module and the record saved, the data in the second and subsequent records is not saved. Any data entered into the first row is saved correctly.	Data entered into the second and subsequent rows is now saved correctly.
If the <i>Parent</i> field links to the current record and the Parts tab is displayed in the Catalogue module, the Windows client will appear to freeze. The problem is caused by a link referring to itself.	The Parent linking code has been modified to check for records that link to themselves. If a record is found, a suitable message is displayed and an empty Parts tab displayed.
When dittoing into a linked field in a LinkGrid control via the Edit>Ditto>Current Field command, the linked module is invoked and a search conducted using the value in the dittoed field. Since the value is being dittoed, the linked module should not be invoked and the value just added to the grid.	The linked module is no longer invoked when dittoing a value into a link field in a LinkGrid control.

Issue	Resolution
Under certain conditions the server-side Lookup List server (lutserver) may delete values from the Lookup List module when the values are still in use. The entries are only deleted if a given value has more than one punctuation variant (e.g. "Brown, Bill" and "Brown - Bill"). Entries should only be deleted if they are no longer used in any records.	The Lookup List server no longer deletes values that are still in use.
The server-side Lookup List server (lutserver) may delete values from the Lookup List module where a Lookup List value is a subset of a longer Lookup List value (e.g. "Damage" and "Paint Damage"). Entries should only be deleted if they are no longer used in any records.	The Lookup List server no longer deletes values that are still in use.
The Calendar pop-up button is still displayed when its associated field is hidden by permissions. The issue only arises in Query mode where the associated control has had the <code>dvQuery</code> permission removed from the Column Access Registry entry.	The Calendar pop-up button is now hidden when its associated field is hidden.
If the Lookup Exact Registry entry is enabled on a field in a hierarchy and a value is selected from the Lookup List in Query mode, the displayed value is enclosed in double quotes. If a search is performed, no matching records will result. The lack of matches is caused by the double quote characters being included in the search.	The double quote characters are removed from the search term when the query is performed for columns that have the Lookup Exact Registry entry enabled.
The Column Access Modifier Registry entry may not be applied correctly when the field whose value is being checked contains a list of values. The issue only arises if there are three or more values in the list.	The Column Access Modifier Registry entry is applied correctly for list fields regardless of the number of entries in the list.
The Add Resource... button on the Multimedia tab in the Multimedia module may become hidden when a user changes to either Medium or Large fonts in the Windows font settings.	The Add Resource... button now displays correctly regardless of the user's Windows font setting.
The error message TexAPI Error: End of file. (Number -18) may occur when the last record in the Multimedia module is deleted while in List View. The error will only occur if the last record had an image associated with it.	The error message no longer occurs when the last record is deleted in List View.

Issue	Resolution
<p>The EMu date and time formats are not applied to values entered into fields created as Admin Task parameters. The current Windows format is used which may cause a mismatch between the value entered and the value expected by the Admin Task handler.</p>	<p>The correct date and time formats are now applied to Admin Task parameters.</p>
<p>Attaching batches of records to the Delete tab in the Operations module via drag and drop from the Catalogue module may be very slow, even for small numbers of records.</p>	<p>The time taken to attach records to the Delete tab has been improved significantly.</p>

Upgrade Notes

The upgrade from EMu Version 4.2 to EMu 4.3 involves a number of steps. Please follow the instructions below carefully.

Do not skip any steps under any circumstances.

Before proceeding with the update please ensure that a complete backup of the EMu server exists and is restorable.

There are four components that require upgrading:

- Texpress (the database engine)
- TexAPI (web services)
- EMu Server (the application)
- EMu Client (the client)

The notes below detail how to upgrade all systems. Check the [Releases](#) table for Client specific notes.

In the notes below, *clientname* refers to the name of the client directory for the current installation. The term `~emu` is used to refer to user `emu`'s home directory. This is normally `/home/emu`.

Stopping EMu services

1. Log in as `emu`
2. Enter `client clientname`
3. Enter `ls -l loads/*/data* local/loads/*/data*`
4. Check that each `data` file is empty and that no `data.t` files exist.
If `data.t` files do exist, please wait for the loads to drain before proceeding.
5. Enter `emuload stop`
6. Enter `emuweb stop`
7. Enter `texlicstatus`
Make sure no one is using the system.
The upgrade will not complete successfully if users are accessing data.

Record Session

Each step in the upgrade process produces detailed output. In most cases this output will exceed the size of the screen. It is strongly recommended that the output of the upgrade session is recorded, so if errors occur, the output can be examined.

1. Enter `script /tmp/output-4-3`

A new shell will start and all output recorded until the shell is terminated.

Installing Texpress

Installing Texpress 8.3 is only required for the first client upgraded to EMu 4.3. Once Texpress 8.3 has been installed, this section may be skipped for subsequent upgrades.

1. Enter `cd ~emu`
2. Enter `mkdir -p texpress/8.3.xxx/install` (where `xxx` is the patch level number).
3. Enter `cd texpress/8.3.xxx/install`
4. Obtain the appropriate [Texpress version](#) for your Unix machine.
Save the release in `~emu/texpress/8.3.xxx/install`, calling it `texpress.sh`.
5. Enter `sh texpress.sh`
The Texpress release will be extracted.
6. Enter `./profile`
7. Enter `bin/texinstall ~emu/texpress/8.3.xxx`
The Texpress installation script will commence.
8. Enter `cd ~emu/texpress/8.3.xxx`
9. Enter `./profile`
10. Enter `bin/texlicinfo`
Obtain your Texpress licence code and place it in a file called `.licence`.
11. Enter `bin/texlicset < .licence` to install the licence.
12. Enter `\rm -fr install`
13. Enter `cd ~emu/texpress`
14. Enter `ln -s 8.3.xxx 8.3`

Upgrading TexAPI

Installing TexAPI is only required for the first client upgraded to EMu 4.3. Once TexAPI has been installed, this section may be skipped for subsequent upgrades.

1. Enter `cd ~emu/texpress`
2. Enter `mkdir 6.0.xxx`
3. Obtain the appropriate [TexAPI version](#) for your Unix machine.
Save the release in `~emu/texpress`, calling it `texapi.sh`.
4. Enter `sh texapi.sh -i ~emu/texpress/6.0.xxx` (expand the `~emu`).
5. Enter `\rm -f texapi`
6. Enter `ln -s 6.0.xxx texapi`
7. Enter `\rm -f texapi.sh`

Upgrading EMu Server

1. Enter `cd ~emu/clientname`
2. Enter `mkdir install`
3. Enter `cd install`
4. Obtain the appropriate [EMu server version bundle](#).
Save the release bundle file in `~emu/clientname/install` calling it `emu.sh`.
5. Enter `sh emu.sh`
The EMu release will be extracted.

6. Enter `./profile`
7. Enter `bin/emuinstall clientname`
The EMu installation script will commence.
8. Enter `cd ~emu/clientname`
9. Enter `cp .profile.parent ../profile`
10. Enter `./profile`
11. Enter `client clientname`
12. Enter `emureindex`
13. Enter `vi etc/config`

Add the following text to the end of the file (if it does not exist already):

```
#
# EMUSERVERPORT is the port the EMu client uses to connect to the
# EMu server.
# The port corresponds to the "Service" value entered in the EMu
# Client Login box.
#
EMUSERVERPORT=port
export EMUSERVERPORT
```

where *port* is the service name used to connect to this EMu server.

Save the file.

14. Enter `EDITOR=vi crontab -e`
Add the following entry to the end of the file:

```
#
# Run Scheduled Operations
#
0 20 * * * ~emu/bin/emurun emuoperations 2>&1 | ~emu/bin/emurun
emulogger -t "KE EMu Scheduled Operations Report" -z operations
```

The start time may need to be varied to fit in with existing maintenance jobs.

15. Removal of the temporary directory (and its contents) is recommended:

Enter `\rm -fr install`

16. Enter `upgrade-4-3`

The client will now be upgraded to EMu 4.3. If you are upgrading from a version prior to EMu 4.3, you must run the upgrade scripts for all versions after the old version before running the EMu 4.3 upgrade.

Starting EMu services

1. Enter `emuload start`
2. Enter `emuload status`
Check that all loads started successfully. Investigate any loads that failed to start.
3. Enter `emuweb start`

Record Session

The recording of the upgrade session may now be terminated.

1. Enter `exit`

The session output is available in /tmp/output-4-3.

Upgrading EMu Client

EMu 4.3 does not require the new Windows client to be installed on every machine for network installations. Updating the network server is sufficient. For standalone installations a new client is required on each machine. To upgrade the EMu Client follow the [Installing EMu Client](#) notes.



EMu Documentation

Scheduled Operations

Document Version 1.1

EMu version 4.3

EMu
Museum
Management
System



kesoftware.com
©2014 KE Software
All rights reserved.

Contents

SECTION 1	Overview	1
SECTION 2	How to schedule an operation	3
	The Operation tab	3
	Delete Operation: the Delete tab	6
	Image Import Operation: the Image Import tab	8
	Merge Operation: the Merge tab	10
	Examples	12
	Scenario 1	12
	Scenario 2	14
SECTION 3	Viewing Operation Results	17
	View Result Files	17
	Save all Result Files	18
	Save a Result File	18
SECTION 4	How to create an additional type of Scheduled Operation	19
	Storage of Scheduled Operations scripts	20
	Invoking a scheduled operation	22
	Accessing information from a Scheduled Operations record	23
	An example operation	25
	Useful functions that may be called from within an operation	30
	OpenLogFile	31
	FileLog	31
	GetStartPosition	32
	AddToProcessed	32
	GetAttachmentFields	33
SECTION 5	emuoperations	35
	Using emuoperations	35
	Configuring emuoperations	36
	Index	37

SECTION 1

Overview



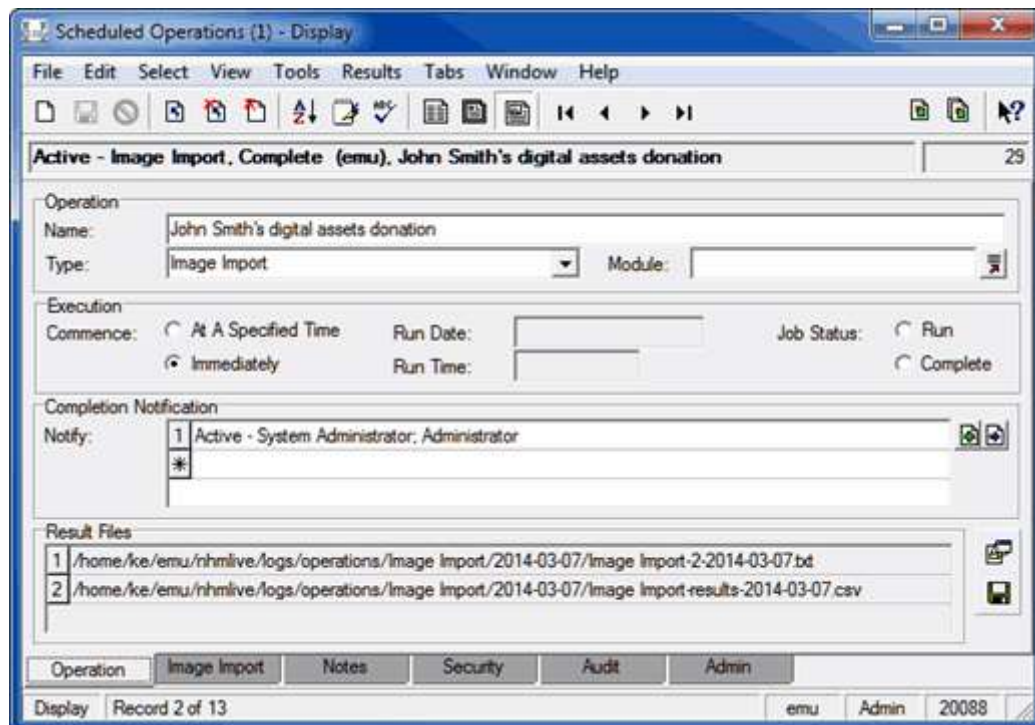
In order to use the Scheduled Operations facility, a user must have (or be a member of a group that has) Table Access to the Operations module (eoperations) and the `daInsert` operations permission.

The Scheduled Operations facility introduced with EMu 4.3 enables the scheduling of operations to be run immediately or at a specified date and time. Operations are scheduled in the Scheduled Operations module, which is accessed by selecting



Operations

in the Command Centre:



With the Scheduled Operations facility it is possible to define:

- The type of operation to run
- The module to apply the operation to
- A time to commence the operation
- People to notify when the operation is complete

A scheduled operation is defined and stored as a record in the Scheduled Operations module.

When a scheduled operation is run, any files created during the operation are listed in the *Result Files* table on the Operation tab. Result files can be viewed and saved.

Audit logs are produced for all scheduled operations, allowing suitably authorised users to search / view the results of all operations performed by all users.

EMu 4.3 supports three types of scheduled operation:

- Merge Records
- Delete Records
- Image Import




System Administrators may define additional types of Scheduled Operation as required. See How to create an additional type of Scheduled Operation (page 19) for details.

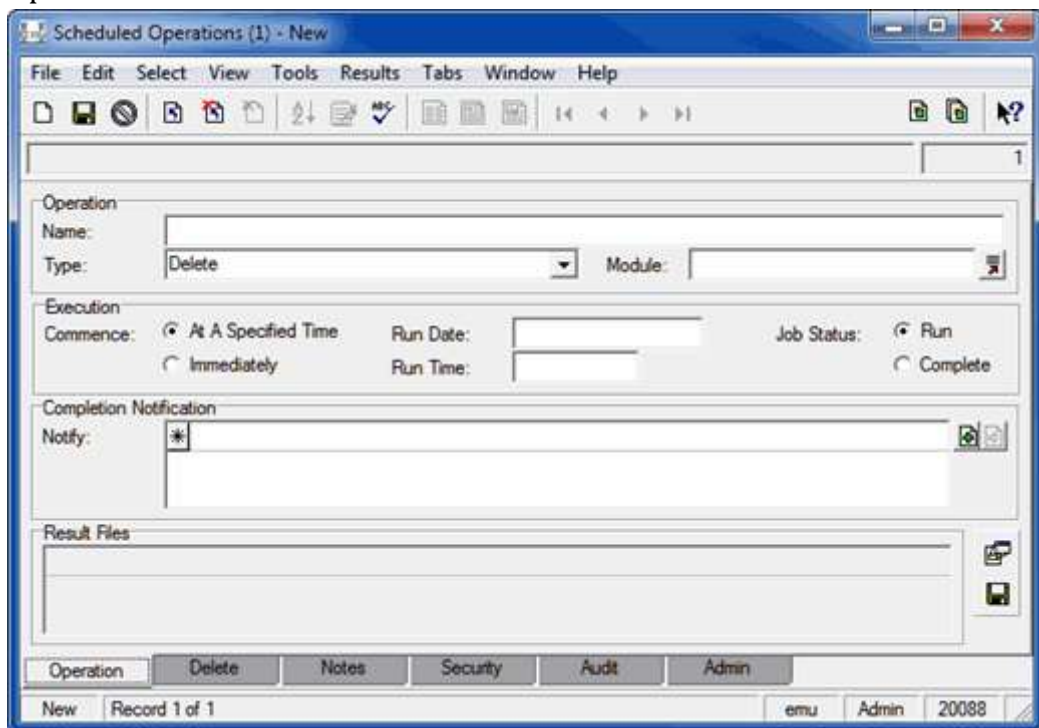
SECTION 2

How to schedule an operation

Scheduling an operation is similar to creating any other record.

The Operation tab

1. Select  **Operations** in the Command Centre to display the Scheduled Operations module:



2. Enter a descriptive name for the operation in the *Name: (Operation)* field.
3. Select the type of operation to be performed from the *Type: (Operation)* drop list. By default, there are three types of operation to choose from:
 - **Delete (page 6)**
Delete a series of IRNs from a module.
 - **Image Import (page 8)**
Import images from a directory into the Multimedia module.
 - **Merge (page 10)**
Merge one or more records with a Target record in a module.



System Administrators may define additional operations as required. See *How to create an additional type of Scheduled Operation* (page 19) for details.

4. In the *Module: (Operation)* field, select the module in which the operation is to be performed.



When scheduling an `Image Import` it is not necessary to specify a module as `emultimedia` (the `Multimedia` module) is implicit to the operation (images are imported into the `emultimedia` table).

5. In the *Execution* group of fields specify the time that the operation will be executed. There are two options:

- `At A Specified Time`

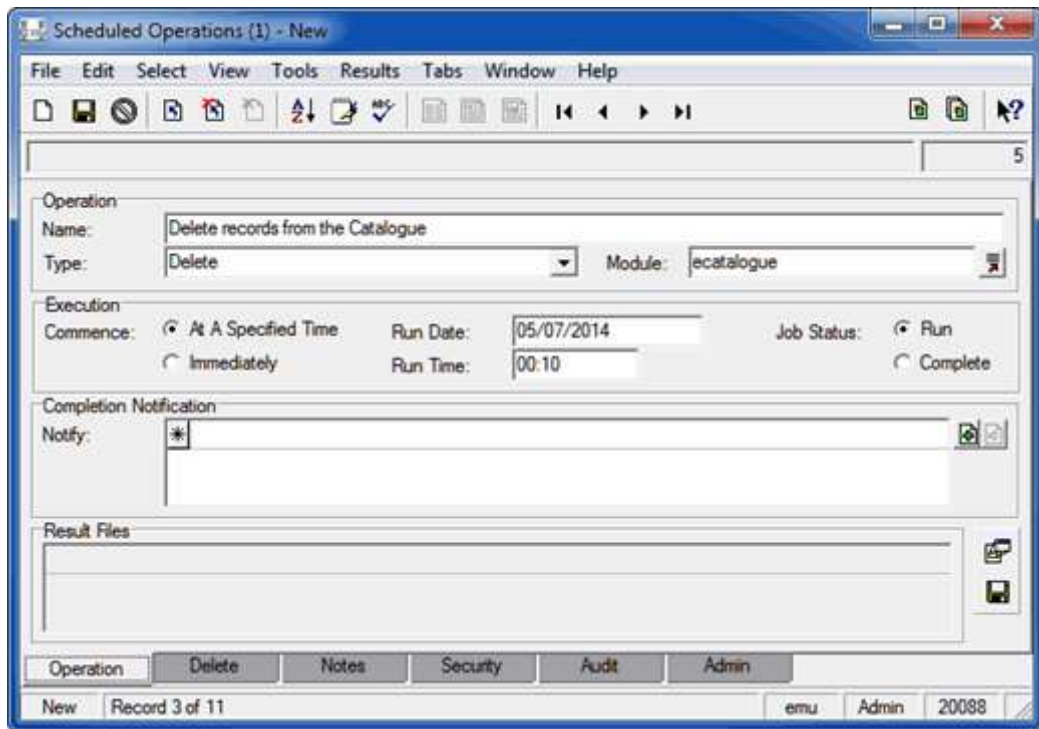
With this option selected it is possible to specify a *Run Date* and *Run Time* for the operation to commence its processing. This allows operations to be run outside of normal business hours or at the weekend.



A date and time specified here is the **earliest** that the operation will be run. The actual time at which an operation is run will depend on when the `emuoperations` script is scheduled to run (page 22): `emuoperations` is the script used to execute an operation that has been scheduled in a record in the `Schedule Operations` module (page 35). When `emuoperations` is run, it looks for any operations that were scheduled to run prior to the current date and time and commences them. Thus, if `emuoperations` is scheduled to run once per day, it will commence any operation scheduled to run in the previous 24 hours (in theory an operation could have been scheduled to run 23 hours and 59 minutes earlier). If `emuoperations` is to be run once per day, it probably makes sense therefore to schedule operations close to the time at which `emuoperations` is run. Alternatively, `emuoperations` can be run at various times throughout the day.

- `Immediately`

With this option the operation will commence as soon as the record is saved.



6. In *Notify: (Completion Notification)* attach the Parties record for anyone who is to be notified by email when the scheduled operation has completed.

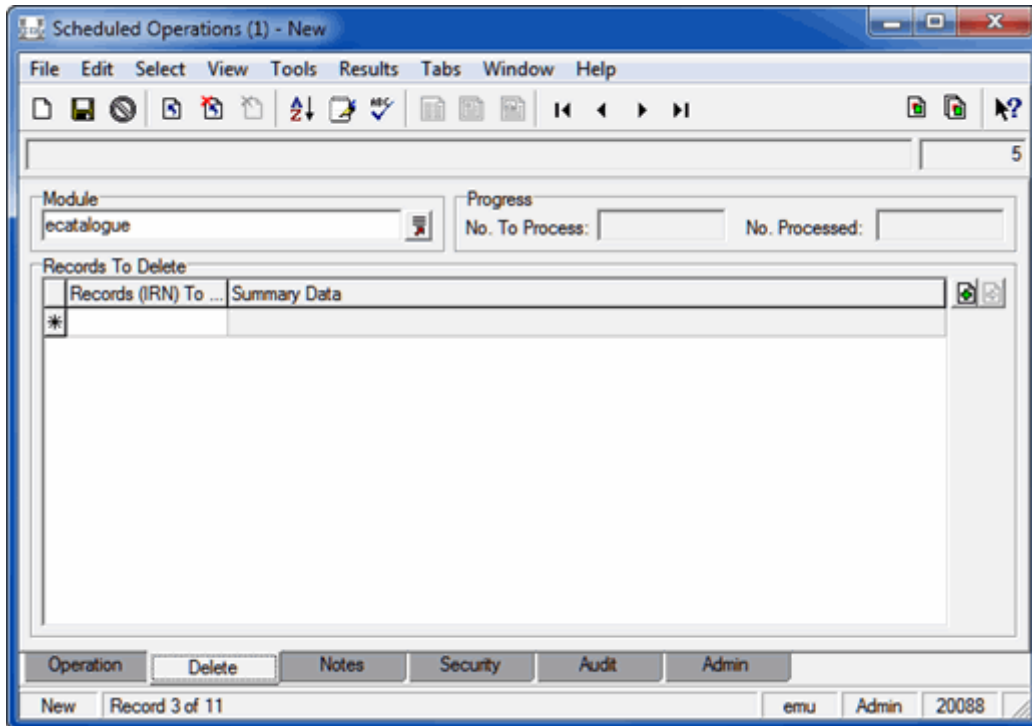





Email notifications will only be received by parties added to the *Notify: (Completion Notification)* table if their Parties record includes a valid email address in the *Email: (Internet Details)* field.

Job Status: (Execution) indicates that the operation is waiting to be run, or that it has been run and is complete. Note that if an operation terminates unexpectedly, the status will remain as *Run* until the operation is restarted and it completes.

Delete Operation: the Delete tab

When **Delete** is selected from the *Type: (Operation)* drop list on the Operation tab, the Delete tab displays:



1. The *Module* field will list the module from which records will be deleted if a module was specified (Step 4) on the Operation tab (page 3).
If a module was not selected on the Operation tab, specify in the *Module* field which module the records are to be deleted from.
2. In the *Records To Delete* table add the records that are to be deleted from the module specified in the *Module* field.
Records can be added through the attachment or drag and drop process:
 - 2.1. Click  beside the *Records To Delete* table to open the module specified in the *Module* field.
 - 2.2. Search the module for the record or records to delete and click **Attach Current Record**  or **Attach Selected Records**  in the Tool bar to add the record(s) to the *Records To Delete* table in the Scheduled Operations module.
-OR-
 - 2.3. Open the module specified in the *Module* field and search for the record or records to be deleted.
 - 2.4. Select the record or records in List View and drag and drop them to the *Records To Delete* table in the Scheduled Operations module.
3. Save the record:

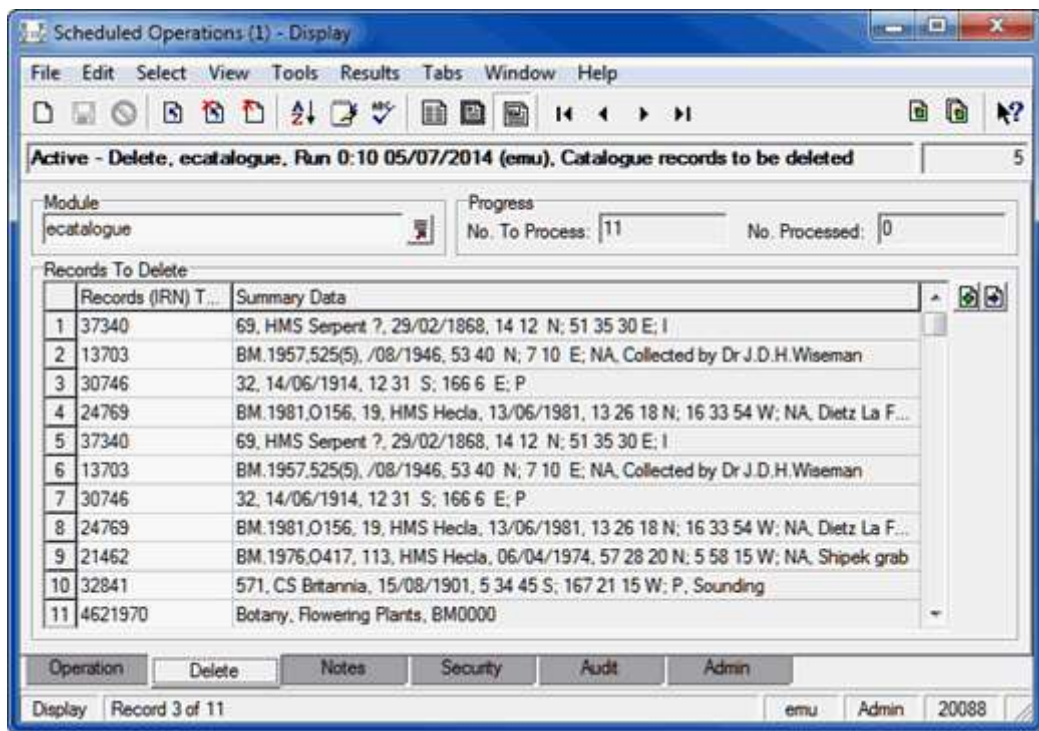
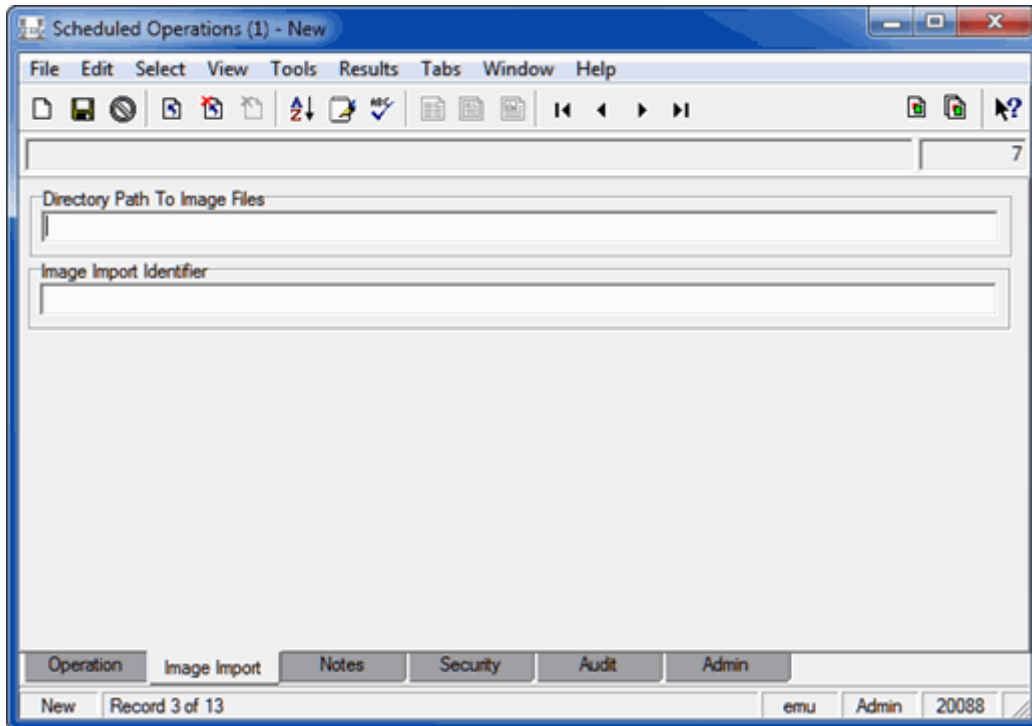
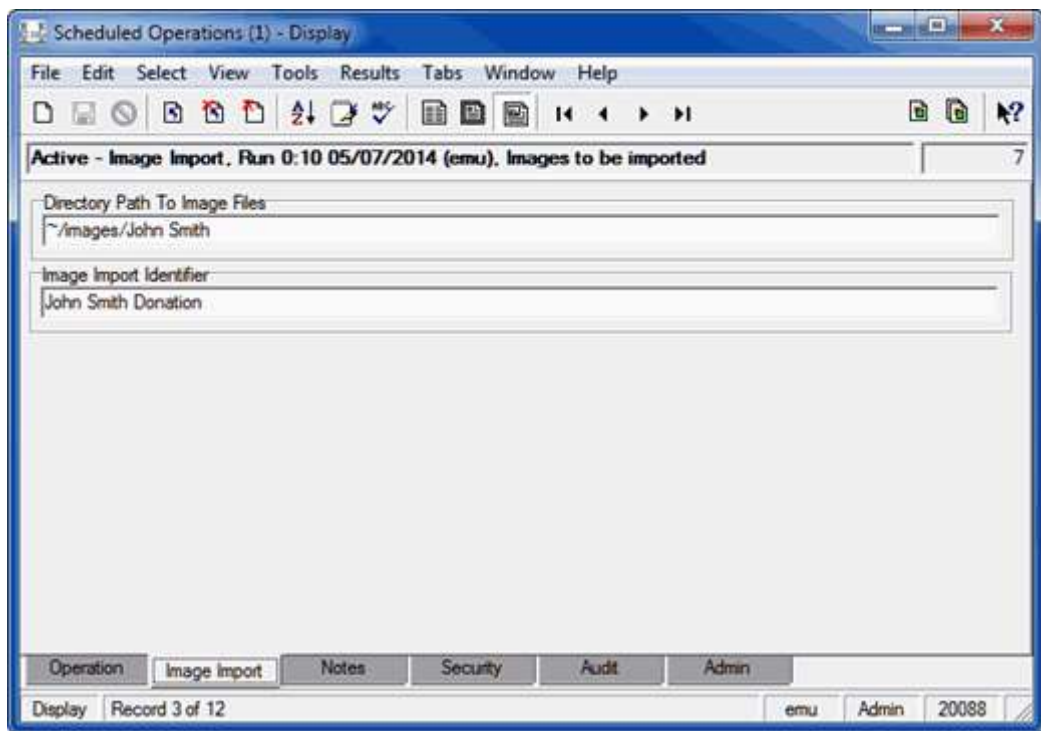


Image Import Operation: the Image Import tab

When **Image Import** is selected from the *Type: (Operation)* drop list on the Operation tab, the Image Import tab displays:

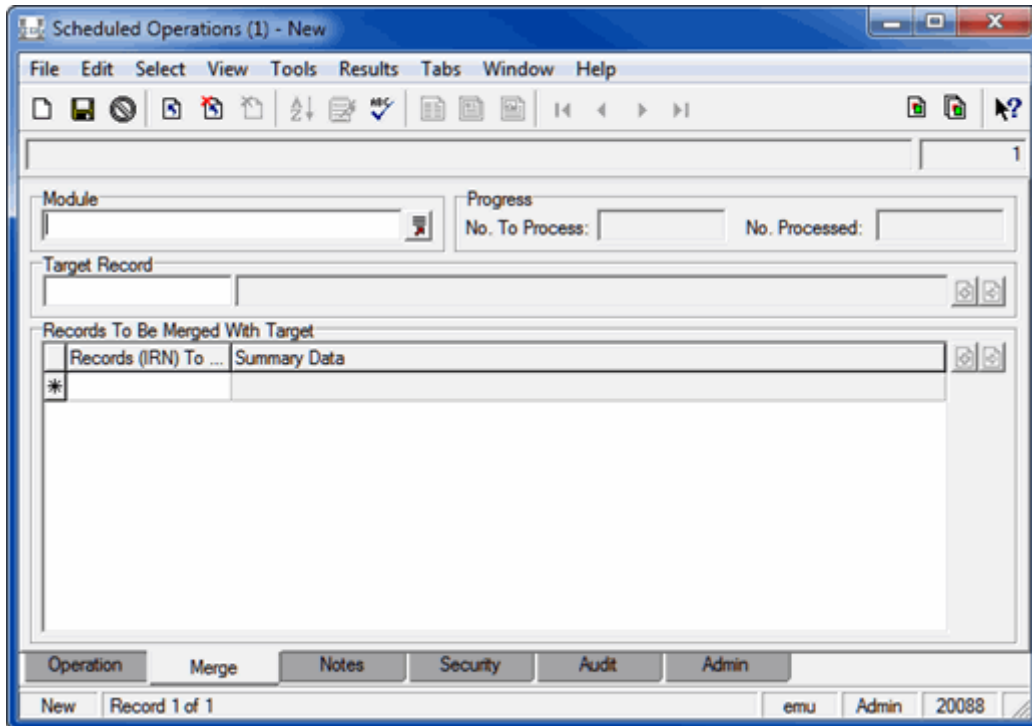




1. In *Directory Path To Image Files*, enter the pathway to the image files to be imported.
The path may be a full path:
/home/emu/..
or a relative path:
~/../.. or ../..
2. If required, enter an identifier in the *Image Import Identifier* field. The value entered here will be stored in the *Import Identifier* field on the Admin tab of all Multimedia records created through this scheduled import.
3. Save the record:




Merge Operation: the Merge tab


When **Merge** is selected from the *Type: (Operation)* drop list on the Operation tab, the Merge tab displays:

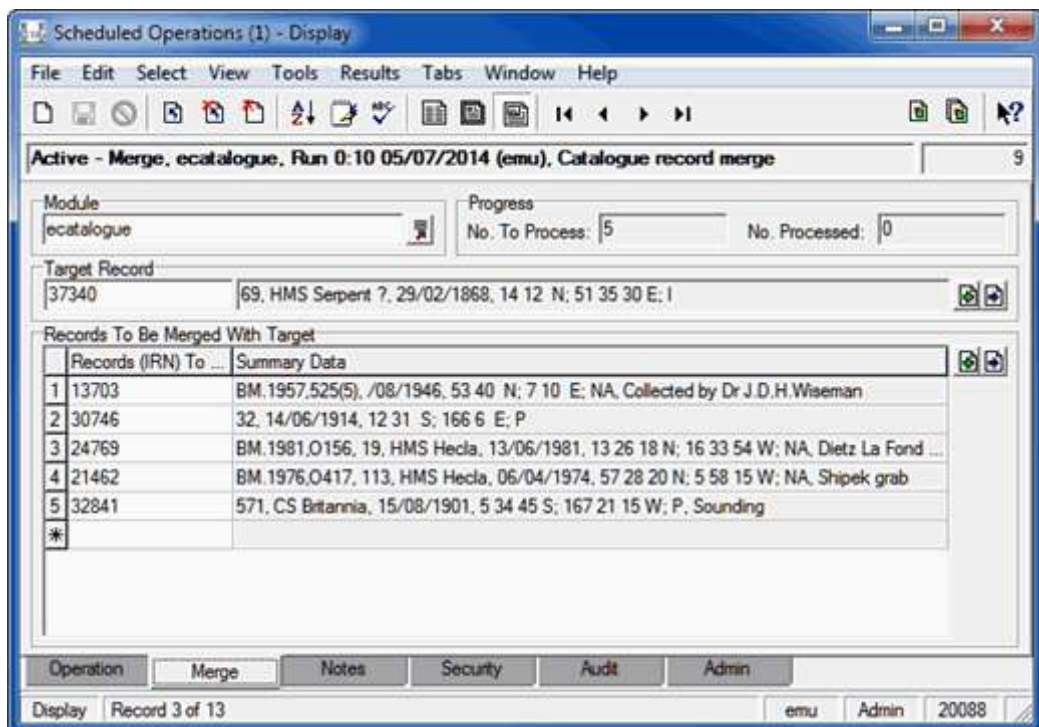


1. The *Module* field will list the module in which records will be merged if a module was specified (Step 4) on the Operation tab (page 3).
If a module was not selected on the Operation tab, specify in the *Module* field in which module the merge will take place.
2. In the *Target Record* field add the record that will be the target of the merge (i.e. the record with which one or more records will be merged).
Records can be added through the attachment or drag and drop process:
 - 2.1. Click  beside the *Target Record* field to open the module specified in the *Module* field.
 - 2.2. Search the module for the Target Record and click **Attach Current Record**  in the Tool bar to add the record to the *Target Record* field in the Scheduled Operations module.

-OR-

 - 2.3. Open the module specified in the *Module* field and search for the Target Record.
 - 2.4. Drag and drop the Target Record to the *Target Record* field in the Scheduled Operations module. There are various ways to do this:
 - In List View click the record to drag and drop it on the *Target Record* field in the Scheduled Operations module.
 - Select the record in List View and drag the Drag Current Record button  in the Tool bar to the *Target Record* field in the

- Scheduled Operations module.
- Display the record in Details View and drag the Drag Current Record button  in the Tool bar to the *Target Record* field in the Scheduled Operations module.
3. In the *Records To Be Merged With Target* table add the records that are to be merged with the Target Record
Records can be added through the attachment or drag and drop process described earlier (page 6).
 4. Save the record:



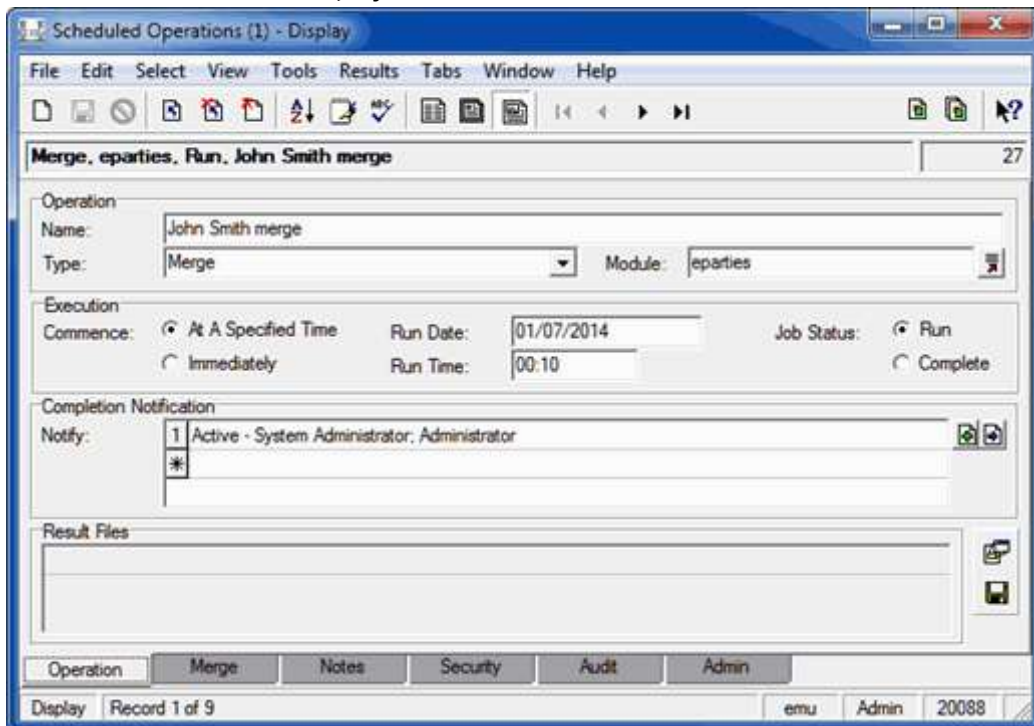
Examples

Scenario 1

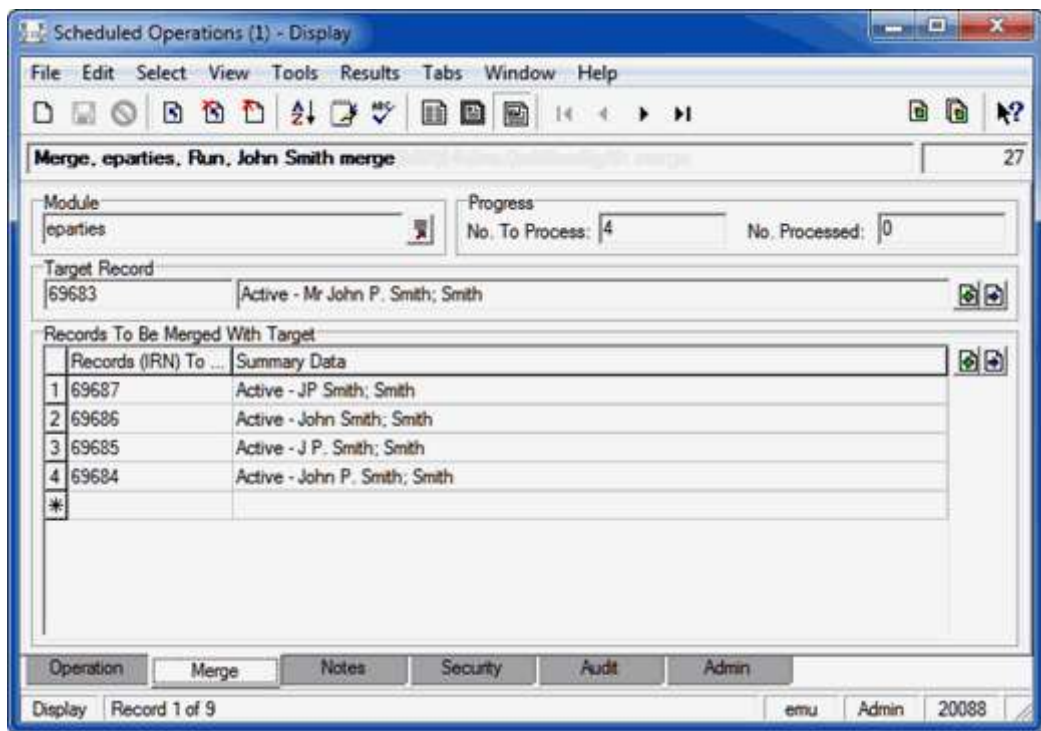
A record clean up project is under way. As part of the clean up we wish to merge five variations of John Smith's Parties record into one. As users are still entering records, we need to wait until 1 July before we can run the Merge.

Solution

1. Add a Scheduled Operations record with a *Type* of Merge for eparties, scheduled to run at 12:10 AM on 1 July:



2. Identify one of the five John Smith Parties records as the Target Record and attach it to the *Target Record* field on the Merge tab of the Scheduled Operation record.
3. Add the remaining four Parties records for John Smith to the *Records To Be Merged With Target* table:

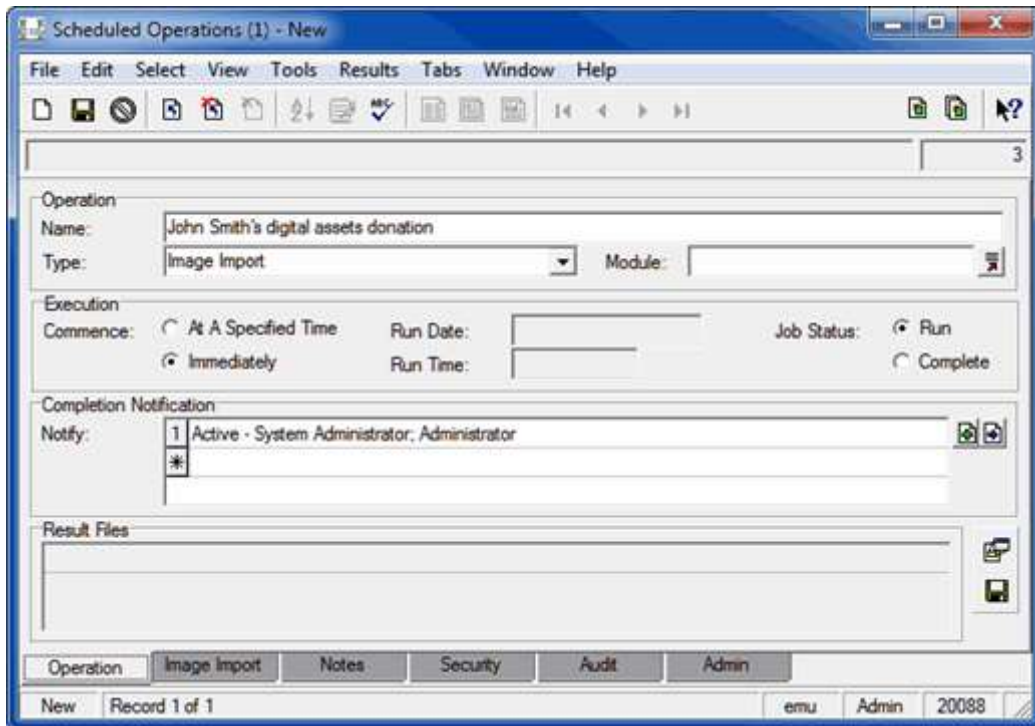


Scenario 2

A large number of digital assets have been donated to your institution. Rather than load them individually, you would like to have them loaded automatically commencing immediately.

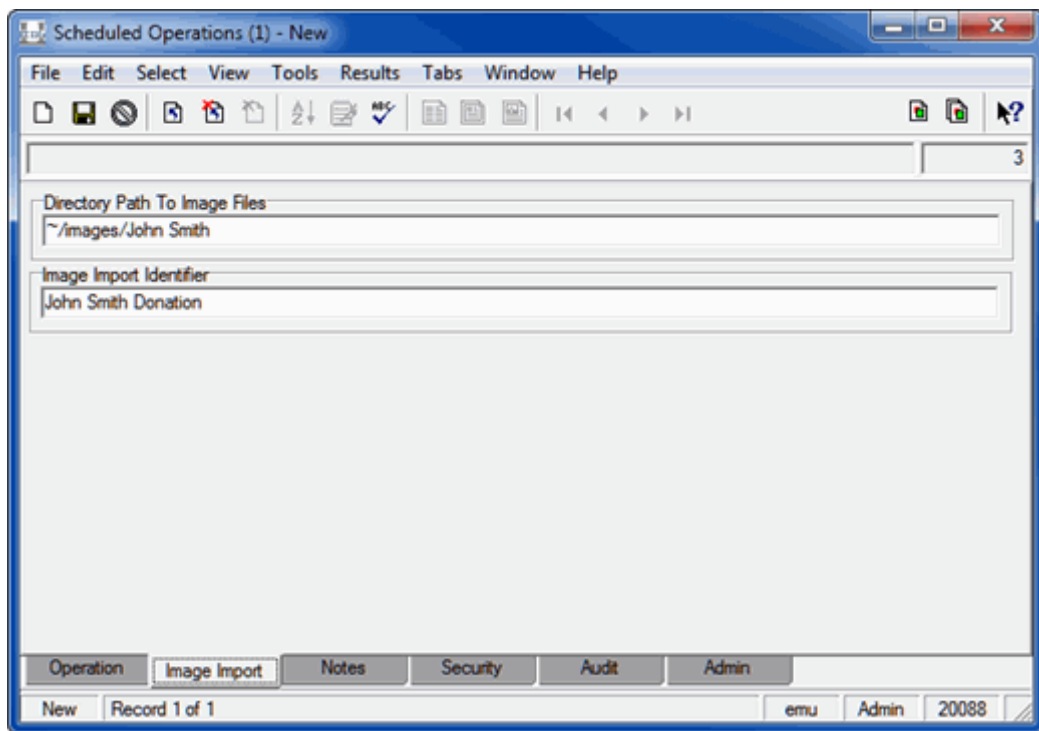
Solution

1. Add a Scheduled Operations record with a *Type* of Image Import to commence loading the digital assets immediately:



When scheduling an Image Import it is not necessary to specify a module as emultimedia (the Multimedia module) is implicit to the operation (images are imported into the emultimedia table).

2. On the Image Import tab specify the directory where the digital assets are stored and an identifier for the created records:

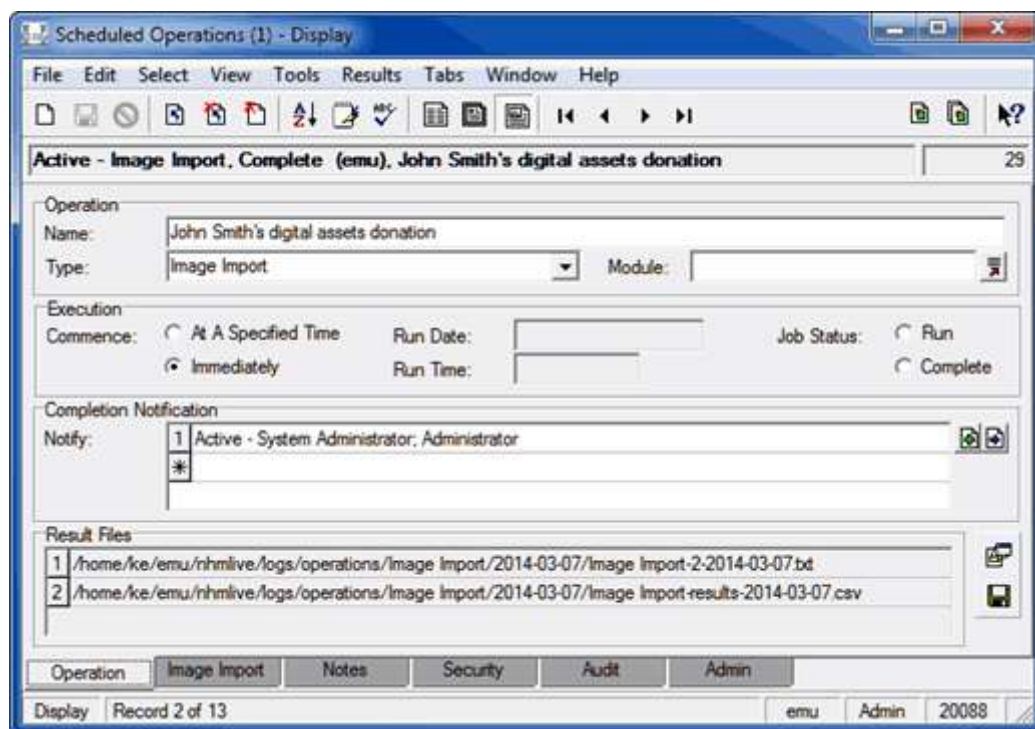


When this record is saved the digital asset import will commence without the need for any further action from the user, who will be able to continue with their other work.

SECTION 3

Viewing Operation Results


Scheduled operations are run automatically by EMu. For each operation executed a Results File is created and added to the *Result Files* table on the Operation tab of the Scheduled Operations record. The files are stored on the EMu server:



View Result Files


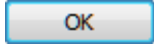
1. Select **Results>Launch Viewer>[Result File]** in the Menu bar.

-OR-

Select the row in the *Result Files* table with the file to be viewed & click .

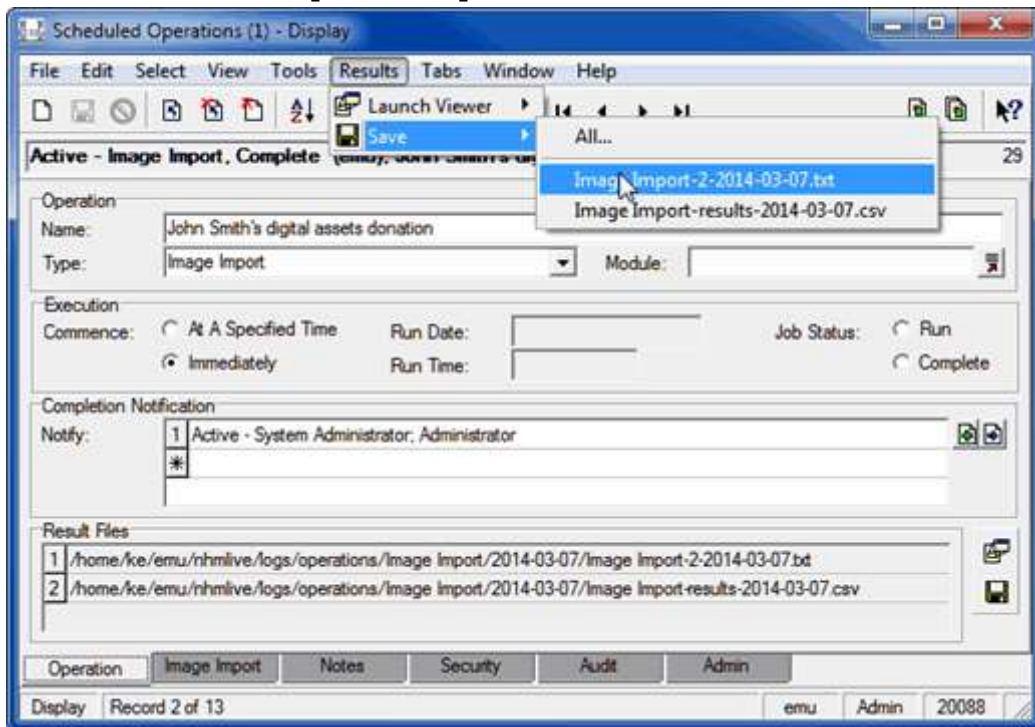
The application / viewer associated with the file extension is invoked to display the file.

Save all Result Files

1. Select  beside the *Result Files* table
-OR-
Select **Results>Save>All** in the Menu bar.
The Browse for Folder dialogue displays.
2. Choose the directory into which all Result Files will be saved.
3. Select .

Save a Result File

1. Select **Results>Save>[Result File]** in the Menu bar:



The Save As dialogue displays.

2. Choose the location to save the Result File and click .

SECTION 4

How to create an additional type of Scheduled Operation

EMu provides three Scheduled Operations functions by default:

- Delete
- Image Import
- Merge

In this section we examine how System Administrators can create an additional type of Scheduled Operation.

Storage of Scheduled Operations scripts

Each type of Scheduled Operation (e.g. Delete, Merge, etc.) is defined by a script which resides under the `etc/operations` or `local/etc/operations` directory on the EMu server.



When adding a script for an additional type of Scheduled Operation for your EMu system, place it under `local/etc/operations` to avoid the risk of having it overwritten during EMu upgrades.

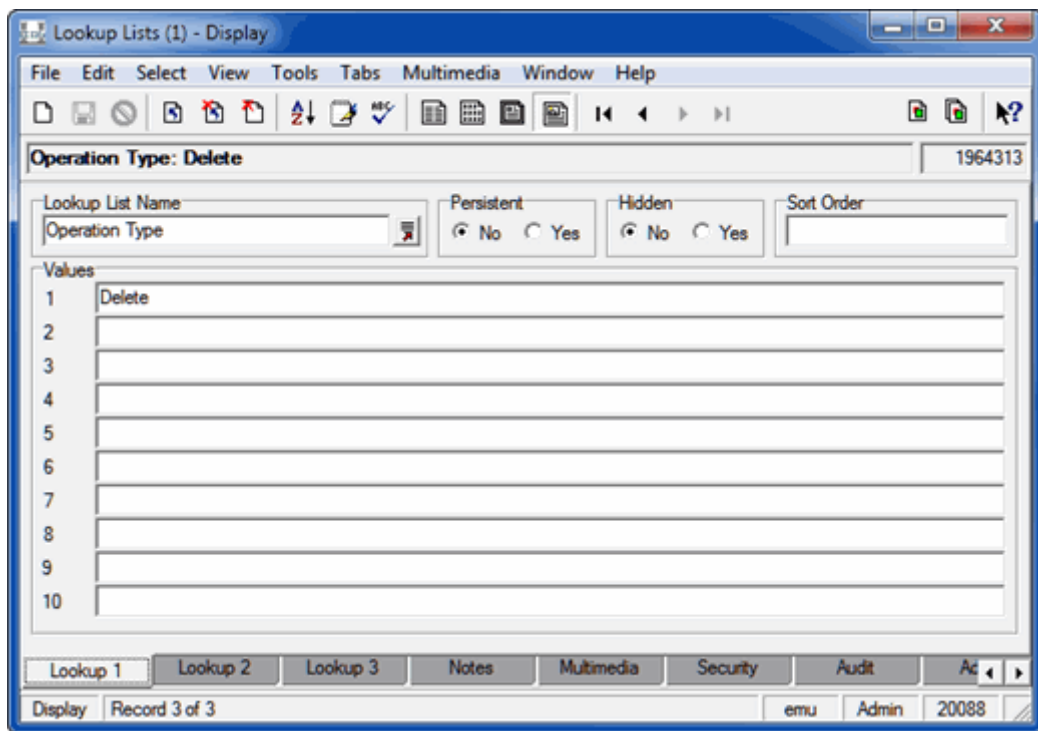
The script includes the name of the operation which will be listed in the *Type: (Operation)* drop list on the Operation tab of the Scheduled Operations module.

When the `emuoperations` process runs it scans the `etc/operations` and `local/etc/operations` directories to locate the scripts for all types of Scheduled Operations (files that end with a `.pl` extension) and registers a name for each type of operation found. The following example registers the `Delete` Scheduled Operation:

```
sub
Register
{
    my $plugins = shift;

    #
    # We handle the "Delete" method.
    #
    $plugins->{"Delete"} = \&Delete;
}
```

When a new type of Scheduled Operation is added to EMu, a Lookup List entry needs to be added to the *Operation Type* Lookup List. For the above example a Lookup List record was added to the *Operation Type* Lookup List with a value of `Delete`:



Invoking a scheduled operation

When scheduling an operation in a record in the Scheduled Operations module, the operation can be scheduled to commence:

- At A Specified Time
- OR-
- Immediately

If *Commence: (Execution)* is set to `Immediately`, the operation will be invoked as soon the Scheduled Operations record is saved. The operation will commence running on the EMu server and control returned to the user to continue with their work.

If *Commence: (Execution)* is set to `At A Specified Time`, the operation will be invoked by the `emuoperations` script on the EMu server at the appropriate time.



The execution of each pending operation consumes a licence in the same way that a user would consume a licence to complete the task. Similarly to users performing tasks, multiple operations can be run simultaneously up to the system licence limit.

The `emuoperations` script is designed to be run from cron with an entry similar to the following:

```
30 17 * * * /home/ke/emu/client/bin/emurun emuoperations 2>&1 |
/home/ke/emu/client/bin/emurun emulogger -t "KE EMu Operations" -z
operations
```

The script is typically run once per day but can be configured to run any number of times during the day. When the `emuoperations` script runs it looks for any operations that were scheduled to run prior to the current date and time and commences them.



A date and time specified in a Scheduled Operations record is thus the **earliest** that the operation will be run. The actual time at which an operation is run will depend on when the `emuoperations` script is scheduled to run (page 22): `emuoperations` is the script used to execute an operation that has been scheduled in a record in the Schedule Operations module (page 35). When `emuoperations` is run, it looks for any operations that were scheduled to run prior to the current date and time and commences them. Thus, if `emuoperations` is scheduled to run once per day, it will commence any operation scheduled to run in the previous 24 hours: in theory an operation could have been scheduled to run 23 hours and 59 minutes earlier. If `emuoperations` is to be run once per day, it probably makes sense therefore to schedule operations close to the time at which `emuoperations` is run. Alternatively, `emuoperations` can be run at various times throughout the day.

`emuoperations` will also re-run any previous operations that did not complete.

Accessing information from a Scheduled Operations record

Each type of Scheduled Operation registers a function that is called to process the operation. For example, the Delete Scheduled Operation is performed by a registered function called `Delete`.

```
sub
Register
{
    my $plugins = shift;

    #
    # We handle the "Delete" method.
    #
    $plugins->{"Delete"} = \&Delete;
}
```

The function is passed two parameters:

- An IMu session which allows access to EMu records to perform the operation.
- A hash of data from a Scheduled Operations record with details about this particular operation (i.e. when, what records are affected, what module, etc.).

```
sub
Delete
{
    my $imusection = shift;
    my $record = shift;

    #
    # Run the "Delete" operation.
    #
    ...
}
```

The list of keys available in the hash are:

<code>Irn</code>	The IRN of a record in the Scheduled Operations module with details about this scheduled operation.
<code>Name</code>	The name of the operation.
<code>Type</code>	The type of operation.
<code>Module</code>	The module the operation is to be performed on.
<code>ActionIrn</code>	The target IRN for the Merge operation.
<code>IrnsToProcess</code>	The list of IRNs that the operation needs to process.
<code>IrnsProcessed</code>	The list of IRNs that the operation has already processed.



Typically this would be an empty list except when an operation failed to complete.

<code>Directory</code>	The directory which contains files / information required by an operation to process.
<code>Identifier</code>	An identifier to add to records updated as part of running the operation.

The values for the keys are accessed through the `$record` parameter, e.g.:

```
$record->{Module}
```

-OR-

```
@{$record->{IrnsToProcess}}
```


An example operation

In this example a list of IRNs is deleted:

```
#!/usr/bin/perl

use strict;
use warnings;
use lib "$ENV{EMUPATH}/utils/imu/lib";
use IMu::Module;

#
# Registration function.
#
no warnings 'redefine';

sub
Register
{
    my $plugins = shift;

    #
    # We handle the "Delete" method.
    #
    $plugins->{"Delete"} = \&Delete;
}

use warnings 'redefine';

#
# The handler for the "Delete" operation
#
sub
Delete
{
    my ($imusection, $record) = @_ ;
    my ($attachments, $start, @deleteirns, $irn, $i);

    #
    # Check that we have the required information
    #
    if (! defined($record->{IrnsToProcess}) ||
        @{$record->{IrnsToProcess}} == 0)
    {
        FileLog("Error: no irns supplied for
deletion");
        return(1);
    }
    elsif (! defined($record->{Module}) or $record-
>{Module} eq "")
    {
        FileLog("Error: delete module is not
defined");
        return(1);
    }
}
```

```

    }

    #
    # Get the other information that we need to process
    #
    $attachments = GetAttachmentFields($record->
{Module});
    @deleteirns = @{$record->{IrnstoProcess}};
    $start = GetStartPosition($record);
    FileLog("Running DELETE plugin for $record->
{Module}");
    FileLog("%d records scheduled for deletion, starting
at position $start", scalar(@deleteirns));

    #
    # Now delete each record in turn
    #
    for ($i = $start; $i < @deleteirns; $i++)
    {
        $irn = $deleteirns[$i];
        FileLog("Deleting irn $irn...");
        last if (! ProcessDeletion($imusection,
$attachments, $irn, $record));
        AddToProcessed($irn);
    }
    return($i != @deleteirns);
}

#
# Do the actual deletion work
#
sub
ProcessDeletion
{
    my ($imusection, $attachments, $irn, $record) = @_;
    my ($table, $colname, $module, @matches, $hits,
%found, $key, $column);

    eval
    {
        %found = ();
        foreach $key (keys %{$attachments})
        {
            #
            # The assignment here is unusual but
            # odd foreach scoping problem after
            # an exception is thrown.
            #
            $table = $key;
            $module = IMu::Module->new($table,
$imusection);
            foreach $column (keys
%{$attachments->{$table}})

```

```

{
    #
    # Find records which match
this irn
    #
    # $colname = $column;
    $hits = $module-
>findTerms([$colname, $irn]);
    next if ($hits <= 0);
    #
    # Add records to found hash
    #
    FileLog("Found $hits matches
for $colname in $table");
    push(@{$found{$table}-
>{$colname}}, GetMatches($module));
}
}
};
if ($?)
{
    FileLog("Error: failed to process $colname
in $table for irn $irn: $?");
    return(0);
}
@matches = keys %found;
if (@matches)
{
    #
    # Log that we cannot delete the record
    #
    FileLog("Unable to delete irn $irn because
it is attached in the following places:");
    foreach $table (@matches)
    {
        foreach $colname (keys
%{$found{$table}})
        {
            FileLog("\tModule: $table,
Column: $colname, Record(s): " . join(", ",
@{$found{$table}->{$colname}}));
        }
    }
}
else
{
    #
    # Delete the record
    #
    DeleteRecord($imusection, $irn, $record);
}
#
# Add irn to processed

```

```

        #
        return(1);
    }

    #
    # Delete the record
    #
    sub
    DeleteRecord
    {
        my ($imusection, $irn, $record) = @_ ;
        my ($module, $hits, $result);

        eval
        {
            $module = IMu::Module->new($record->
>{Module}, $imusection);
            $hits = $module->findKey($irn);
            if ($hits > 0)
            {
                $result = $module->remove("start",
0, 1);

                if ($result == 0)
                {
                    FileLog("Failed to delete
irn $irn from $record->{Module}");
                }
            }
            else
            {
                FileLog("Failed to find irn $irn in
$record->{Module}");
            }
        };
        if ($?)
        {
            FileLog("Failed to delete $irn from $record->
>{Module}: $?");
        }
    }

    #
    # Get all the records that match the attachment query
    #
    sub
    GetMatches
    {
        my ($module) = @_ ;
        my ($result, @matches, $row);

        #
        # Get all of the records at once
        #

```

```
@matches = ();
$result = $module->fetch("start", 0, -1, "irn");
if ($result->{count})
{
    #
    # Get the irn for each row and push it to
the list of matches
    #
    foreach $row (@{$result->{rows}})
    {
        push(@matches, $row->{irn});
    }
}
return(@matches);
}

1;
```

Useful functions that may be called from within an operation

The following functions are available to be called for use within an operation:

<code>OpenLogFile</code> (page 31)	Opens a results log file and adds it to the list of Result Files.
<code>FileLog</code> (page 31)	Writes a message to the standard operation Result File.
<code>GetStartPosition</code> (page 32)	Determines from what position to start processing the <code>IrnsToProcess</code> list.
<code>AddToProcessed</code> (page 32)	Adds the processed IRN to the <code>IrnsProcessed</code> list.
<code>GetAttachmentFields</code> (page 33)	Returns a hash of all attachment fields for a module.

OpenLogFile

Input parameters: Filename

Returns: File Handle for writing and an indication if the file already exists
(from a previous attempt to run the operation)

```
sub
DoSomething
{
    my $handle;
    my $exists;

    #
    # Open a file for logging results.
    #
    ($handle, $exists) = OpenLogFile("results.csv");
    if ($exists)
    {
        print $handle "...Resuming processing...";
    }
    ...
    close($handle);
}
```

FileLog

Input parameters: Format string and parameters

Returns: Nothing

```
sub
DoSomething
{

    #
    # Log a message.
    #
    FileLog("Error: no irns supplied for deletion");
    ...
    #
    # Log a formatted message.
    #
    FileLog("%d records scheduled for deletion, starting
at position $start", scalar(@deleteirns));
}
```

GetStartPosition

Input parameters: Record hash passed to operation

Returns: Index into `IrnsToProcess`

```
sub
Operation
{
    my $imusection = shift;
    my $record = shift;
    my $start;

    #
    # Get the start position for processing the
operation.
    #
    $start = GetStartPosition($record);
    ...
}
```

AddToProcessed

Input parameters: IRN

Returns: Nothing

```
sub
Operation
{
    my $imusection = shift;
    my $record = shift;
    my $irn;

    ...

    #
    # Finished processing the operation on an irn.
    #
    AddToProcessed($irn);
    ...
}
```


GetAttachmentFields

Input parameters: Module

Returns: A hash of modules with attachment columns to the requested module

```

sub
Operation
{
    my $imusection = shift;
    my $record = shift;
    my $attachments;
    my $module;
    my $column;

    ...

    #
    # Get the attachment fields for the operation
module.
    #
    $attachments = GetAttachmentFields($record->
>{Module});
    ...

    #
    # Process the attachment fields.
    #
    foreach $module (keys %{$attachments})
    {
        foreach $column (keys %{$attachments->
>{$module}})
        {
            ...
        }
    }
    ...
}

```

SECTION 5

emuoperations

`emuoperations` is a script used to execute scheduled operations.



A date and time specified in a Scheduled Operations record is the **earliest** that the operation will be run. The actual time at which an operation is run will depend on when the `emuoperations` script is scheduled to run (page 22). When run, `emuoperations` looks for any operations that were scheduled to run prior to the current date and time and commences them. Thus, if `emuoperations` is scheduled to run once per day, it will commence any operation scheduled to run in the previous 24 hours (in theory an operation could have been scheduled to run 23 hours and 59 minutes earlier). If `emuoperations` is to be run once per day, it probably makes sense therefore to schedule operations close to the time at which `emuoperations` is run. Alternatively, `emuoperations` can be run at various times throughout the day.

Using emuoperations

`emuoperations` may be used in two ways:

1. **Run all Scheduled Operations**

Usage: `emuoperations`

Any Scheduled Operations required to be run will be executed. The current date and time is used to determine what operations are required. This form of the command is used by cron on a daily basis to ensure all Scheduled Operations for the given day are performed.

2. **Run a specific Scheduled Operation**

Usage: `emuoperations -irn`

The `irn` argument is the IRN of a Scheduled Operations record to be executed. This form of `emuoperations` is useful for testing new operations as it allows a specific operation to be run without waiting for the Scheduled Operations date and time to arrive.

Configuring emuoperations

The `emuoperations` script connects to an `imuserver` to perform the scheduled operations. This connection needs to be made on a specific port. By default the standard EMu configuration port for IMu is the port number 20,000 greater than EMu's client connection port. For example, if the standard EMu client connection port is 20000, the standard `imuserver` connection port is 40000.

The `emuoperations imuserver` must run on a different port to perform the scheduled operations. The `eoperations` load starts the `imuserver` for handling operation requests. The port for `emuoperations` to connect on is defined by the `EMUSERVERPORT` environment variable plus 30000. `EMUSERVERPORT` is the port the EMu client uses to connect to the EMu server and corresponds to the Service value entered in the EMu Client login box.

It is recommended that the Administrator sets the `EMUSERVERPORT` environment variable in the `etc/config` file on the EMu server. Add the following text to the end of the `etc/config` file (if it does not exist already):

```
#
# EMUSERVERPORT is the port the EMu client uses to connect to the
# EMu server.
# The port corresponds to the "Service" value entered in the EMu
# Client Login box.

EMUSERVERPORT=port
export EMUSERVERPORT
```

where *port* is the service name used to connect to this EMu server.

Index

A

Accessing information from a Scheduled Operations record • 23

AddToProcessed • 30, 32

An example operation • 25

C

Configuring emuoperations • 36

D

Delete Operation
the Delete tab • 3, 6, 11

E

emuoperations • 4, 22, 35

Examples • 12

F

FileLog • 30, 31

G

GetAttachmentFields • 30, 33

GetStartPosition • 30, 32

H

How to create an additional type of Scheduled Operation
• 2, 4, 19

How to schedule an operation • 3

I

Image Import Operation
the Image Import tab • 3, 8

Invoking a scheduled operation • 4, 22, 35

M

Merge Operation
the Merge tab • 3, 10

O

OpenLogFile • 30, 31

Overview • 1

S

Save a Result File • 18

Save all Result Files • 18

Scenario 1 • 12

Scenario 2 • 14

Storage of Scheduled Operations scripts • 20

T

The Operation tab • 3, 6, 10

U

Useful functions that may be called from within an operation • 30

Using emuoperations • 35

V

View Result Files • 17

Viewing Operation Results • 17



EMu Documentation

EMu GUID Support

Document Version 1

EMu version 4.3

EMu
Museum
Management
System



kesoftware.com
©2014 KE Software
All rights reserved.

Contents

SECTION 1	GUID Support	1
	Storage of GUIDs in EMu modules	2
	Auto-generation of GUID on record save	4
	Considerations when enabling GUID support	5
	GUID Registry entries	6
	GUID Enabled Registry entry	7
	GUID Auto Types Registry entry	9
	Index	13

SECTION 1

GUID Support

A Globally Unique Identifier (GUID) is a persistent unique reference number used as an identifier in computer software. The term GUID typically refers to various implementations of the Universally Unique Identifier (UUID) standard but is often more generally used to refer to other unique identification methods. Comprehensive details about UUIDs, such as how they are stored (typically as 128-bit values, commonly displayed as 32 hexadecimal digits with groups separated by hyphens) and how they are generated can be found on [Wikipedia](#).

With increasing global initiatives in data sharing, the need for a unique identifier for each discrete bit of data is increasingly important. Already organisations such as the US National Science Foundation (NSF) mandate the use of GUIDs for those wishing to participate in its programs.

GUID support is being implemented in EMu in four phases:

1. Storage of GUIDs in EMu modules
2. GUID generation on EMu record save
3. Local IMu web service for local resolution of EMu GUIDs
4. Global IMu web service portal for global resolution of EMu GUIDs

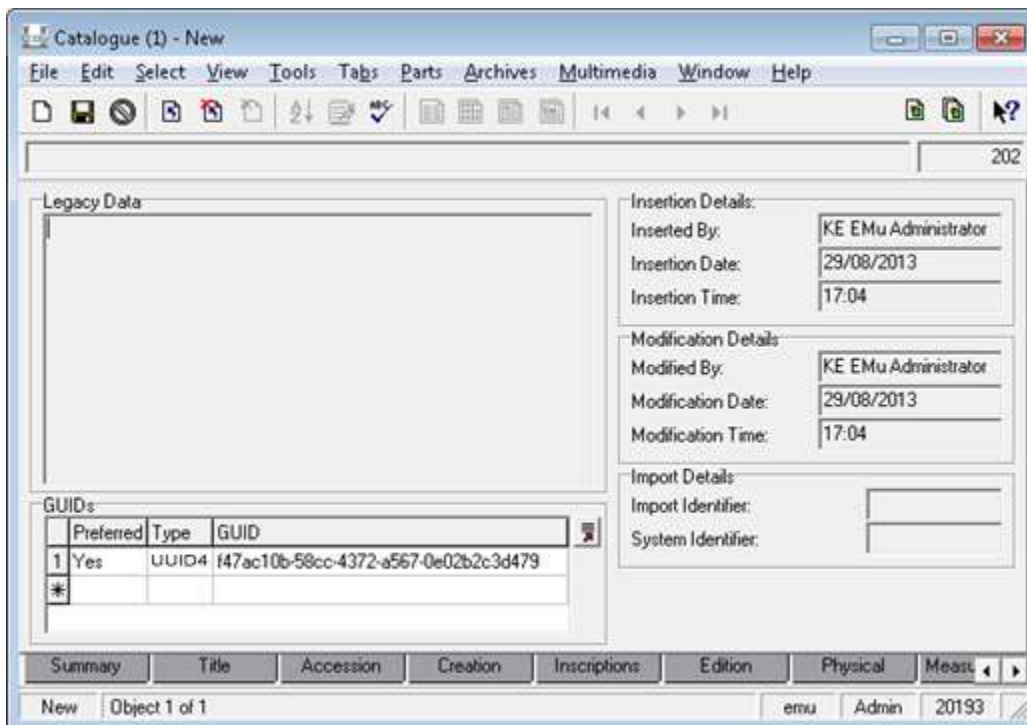
As of July 2014 the first two phases have been completed.

Storage of GUIDs in EMu modules

Almost all EMu modules are capable of making use of GUIDs. Exceptions include:

- Audit Trails (eaudit)
- Condition Checks (econdition)
- Field Help (efieldhelp)
- Gazetteer (egazetteer)
- Internal Movements (einternal)
- Scheduled Operations (eoperations)
- Registry (eregistry)
- Statistics (estatistics)
- Valuations (evaluations)

where the use of GUIDs is thought to be unnecessary. For all GUID capable modules a *GUIDs* table displays on the Admin tab, allowing GUIDs to be added, edited, displayed, searched and included in reports:

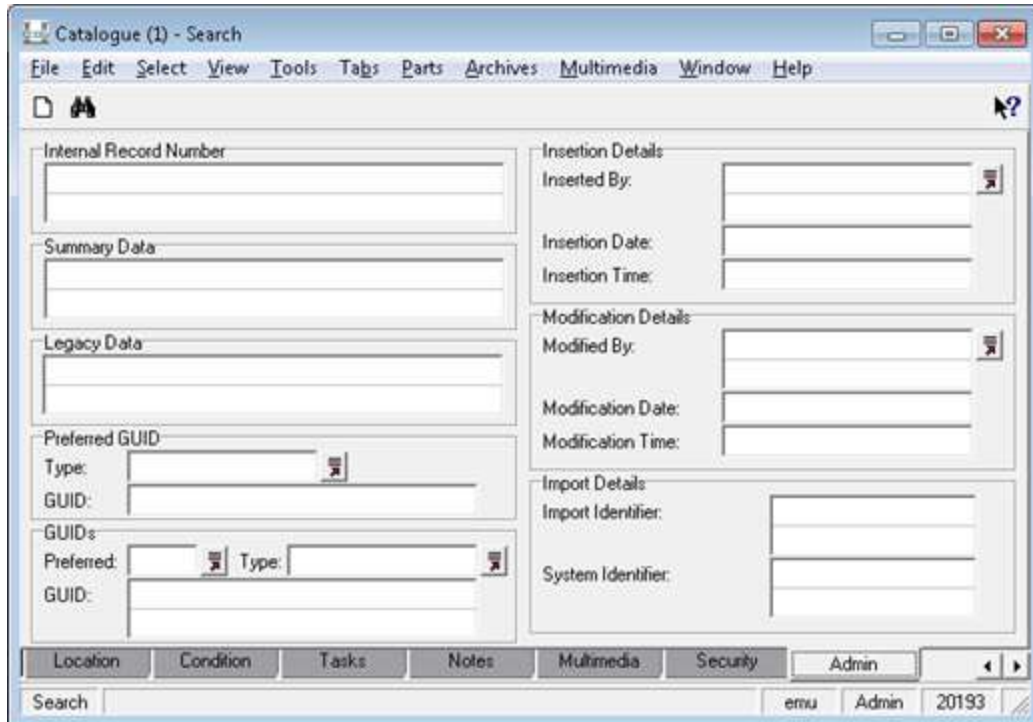


The *GUIDs* table comprises three columns:

Field	Value
<i>Preferred</i>	Yes / No. Only one GUID in the <i>GUIDs</i> table can be marked as preferred.
<i>Type</i>	Lookup List of GUID types.
<i>GUID</i>	The GUID itself.

The *GUIDs* table can hold multiple internally generated and externally generated identifiers.

The GUID fields are also available for querying in Search mode:



The screenshot shows a software window titled "Catalogue (1) - Search" with a menu bar (File, Edit, Select, View, Tools, Tabs, Parts, Archives, Multimedia, Window, Help) and a toolbar. The main area contains several input fields and sections:

- Internal Record Number:** A text input field.
- Summary Data:** A text input field.
- Legacy Data:** A text input field.
- Preferred GUID:** A section with a "Type:" dropdown and a "GUID:" text input field.
- GUIDs:** A section with a "Preferred:" dropdown, a "Type:" dropdown, and a "GUID:" text input field.
- Insertion Details:** A section with "Inserted By:" (dropdown), "Insertion Date:" (text), and "Insertion Time:" (text) fields.
- Modification Details:** A section with "Modified By:" (dropdown), "Modification Date:" (text), and "Modification Time:" (text) fields.
- Import Details:** A section with "Import Identifier:" (text) and "System Identifier:" (text) fields.

At the bottom, there is a navigation bar with tabs: Location, Condition, Tasks, Notes, Multimedia, Security, Admin. Below the tabs is a search bar with the text "Search" and a status bar showing "emu Admin 20193".

Auto-generation of GUID on record save

EMu can auto-generate GUID values on record save. Currently, GUIDs are auto-generated in compliance with UUID Version 4 (page 11).



EMu's GUID support is extensible and it is possible for organisations to substitute or extend the base EMu code to generate other or additional GUIDs. Please contact KE Support for details.

By default, GUID auto-generation is disabled for all modules.

In order to enable auto-generation of GUIDs in a module (or system-wide), the two GUID Registry entries must be specified:

- GUID Enabled Registry entry (page 7) must be set to `true`
-AND-
- A GUID Auto Types Registry entry (page 9) must specify which GUID type(s) to generate for a module (or system-wide). Currently only `UUID4` is supported.

On record save EMu checks whether the *GUIDs* table includes an entry for each type specified in a GUID Auto Types Registry entry, and generates a GUID for each type that is missing.

Currently EMu will auto-generate a UUID Version 4 GUID if one has not already been added to the record's *GUIDs* table.

Considerations when enabling GUID support

- Organisations may need to consider their policy for deleting records that contain locally generated GUIDs. Once a record is assigned a GUID, that record should generally never be deleted. KE's preference has always been that records are `Retired` (using Record Level Security settings) rather than deleted. A `Retired` record is hidden from all users except for those authorised to view it.
- The merging of records that contain GUIDs is another case to be considered. Organisations may want to retain the old GUID(s) from the merged record(s) within the `GUIDs` table on the master record (i.e. the one into which the merge took place).

GUID Registry entries

By default, GUID auto-generation is disabled for all modules.

In order to enable auto-generation of GUIDs in a module (or system-wide), the two GUID Registry entries must be specified:

- GUID Enabled Registry entry (page 7) must be set to `true`
-AND-
- GUID Auto Types Registry entry (page 9) must specify which GUID type(s) to generate. Currently only `UUID4` is supported.

On record save EMu checks whether the *GUIDs* table includes an entry for each type specified in the GUID Auto Types Registry entry, and generates a GUID for each type that is missing.

Currently EMu will auto-generate a UUID Version 4 GUID if one has not already been added to the record's *GUIDs* table.

GUID Enabled Registry entry

Registry Entry	Purpose
GUID Enabled	Specifies whether or not auto-generation of GUIDs is enabled for a table or system-wide.

Overview

A Globally Unique Identifier (GUID) is a persistent unique reference number used as an identifier in computer software. Almost all EMu modules are capable of making use of GUIDs. Exceptions include:

- Audit Trails (eaudit)
- Condition Checks (econdition)
- Field Help (efieldhelp)
- Gazetteer (egazetteer)
- Internal Movements (einternal)
- Scheduled Operations (eoperations)
- Registry (eregistry)
- Statistics (estatistics)
- Valuations (evaluations)

where the use of GUIDs is thought to be unnecessary.

By default auto-generation of GUIDs is disabled. With this Registry entry, it is possible to enable GUID support on a per table or system-wide basis.



In order to enable auto-generation of GUIDs on record save, a GUID Auto Types Registry entry (page 9) must also specify which GUID type(s) to generate. Currently only `UUID4` is supported.

Format of the Registry entry

The format of this Registry entry is:

```
System|Setting|Table|table|GUID Enabled|boolean
System|Setting|Table|Default|GUID Enabled|boolean
```

where:

boolean is `true` (auto-generation of GUIDs is enabled) or `false` (auto-generation of GUIDs is disabled).



If this entry is not present, a setting of `false` is assumed.

Example

This entry specifies that auto-generation of GUIDs is enabled for the Catalogue module:

Field	Value
<i>Key 1</i>	System
<i>Key 2</i>	Setting
<i>Key 3</i>	Table
<i>Key 4</i>	ecatalogue
<i>Key 5</i>	GUID Enabled
<i>Value</i>	true

GUID Auto Types Registry entry

Registry Entry	Purpose
GUID Auto Types	Specifies which GUID variant is used when auto-generating a GUID on record save. Auto-generation of GUIDs also requires the GUID Enabled Registry entry (page 7) to be set to <code>true</code> .

Overview

A Globally Unique Identifier (GUID) is a persistent unique reference number used as an identifier in computer software. Almost all EMu modules are capable of making use of GUIDs. Exceptions include:

- Audit Trails (eaudit)
- Condition Checks (econdition)
- Field Help (efieldhelp)
- Gazetteer (egazetteer)
- Internal Movements (einternal)
- Scheduled Operations (eoperations)
- Registry (eregistry)
- Statistics (estatistics)
- Valuations (evaluations)

where the use of GUIDs is thought to be unnecessary.

EMu can auto-generate GUID values on record save. Currently, GUIDs are auto-generated in compliance with UUID Version 4 (page 11).



EMu's GUID support is extensible and it is possible for organisations to substitute or extend the base EMu code to generate other or additional GUIDs. Please contact KE Support for details.

By default, GUID auto-generation is disabled for all modules.

In order to enable auto-generation of GUIDs in a module (or system-wide), the two GUID Registry entries must be specified:

- GUID Enabled Registry entry (page 7) must be set to `true`
- AND-
- A GUID Auto Types Registry entry must specify which GUID type(s) to generate. Currently only `UUID4` is supported.

On record save EMu checks whether the *GUIDs* table includes an entry for each type specified in the GUID Auto Types Registry entry, and generates a GUID for each type that is missing.

Currently EMu will auto-generate a UUID Version 4 GUID if one has not already been added to the record's *GUIDs* table.

Format of the Registry entry

The format of this Registry entry is:

```
System|Setting|Table|table|GUID Auto Types|type;type;...  
System|Setting|Table|Default|GUID Auto Types|type;type;...
```

where:

type;type;... is a semicolon separated list of GUID variants.



Currently only UUID Version 4 is supported.

Example

This entry specifies that when a GUID is auto-generated on record save in all available tables, the GUID generated will comply with the UUID Version 4 variant:

Field	Value
<i>Key 1</i>	System
<i>Key 2</i>	Setting
<i>Key 3</i>	Table
<i>Key 4</i>	Default
<i>Key 5</i>	GUID Auto Types
<i>Value</i>	UUID4

UUID Version 4 GUID

The GUID Enabled Registry entry (page 7) specifies whether GUID support is enabled for a table or system-wide (by default GUID support is disabled). The GUID Auto Types Registry entry (page 9) specifies which GUID formats are to be auto-generated. In theory more than one variant can be used to specify the auto-generated GUIDs but EMu currently supports only UUID Version 4.

For more details see:

- http://en.wikipedia.org/wiki/Universally_unique_identifier

A UUID is a 128 bit quantity typically represented in text as a 32 character hexadecimal string with hyphen separators at set positions. For example:

```
84b567d5-dbbd-468a-be12-770747ebc397
```

```
71820e9a-1eb0-4b02-9382-20de614fbcdc
```

UUID Version 4 is an extensively used GUID generation method that provides a randomly generated GUID with infinitesimal probability of a duplicate. A prime advantage of this scheme is that GUIDs can be generated quickly and locally on demand without contacting a central naming authority or having to pre-allocate identifiers.

A UUID can be represented simply as a Uniform Resource Name (URN). For example:

```
urn:uuid:84b567d5-dbbd-468a-be12-770747ebc397
```

```
urn:uuid:71820e9a-1eb0-4b02-9382-20de614fbcdc
```

The URN name space identifier of uuid is used.

Refer to:

- http://en.wikipedia.org/wiki/Uniform_resource_identifier
- http://en.wikipedia.org/wiki/Uniform_resource_name
- http://en.wikipedia.org/wiki/Uniform_resource_locator

Index

A

Auto-generation of GUID on record save • 4

C

Considerations when enabling GUID support • 5

G

GUID Auto Types Registry entry • 4, 6, 7, 9, 11

GUID Enabled Registry entry • 4, 6, 7, 9, 11

GUID Registry entries • 6

GUID Support • 1

S

Storage of GUIDs in EMu modules • 2

U

UUID Version 4 GUID • 4, 9, 11